TOKENPASSER
A PETRI NET SPECIFICATION TOOL

NAGW-1333

by

Michael Mittmann

Rensselaer Polytechnic Institute
Electrical, Computer, and Systems Engineering
Troy, New York 12180-3590

January 1991

CIRSSE REPORT #81

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to thank Professor George Saridis the guidance and support given to me during my research. I would also like to thank Dr. Fei-Yue Wang for the tremendous amount of help developing and clarifying this thesis. Finally I would like to thank Mr. Krishnamoorthy for his help with displaying Petri nets on Xwindows.

ABSTRACT

In computer program design it is essential to know the effectiveness of different design options in improving performance, and dependability. This paper provides a description of a CAD tool for distributed hierarchical Petri nets.

After a brief review of Petri nets, Petri net languages, and Petri net transducers, and descriptions of several current Petri net tools, the specifications and design of the TokenPasser tool are presented. TokenPasser is a tool to allow design of distributed hierarchical systems based on Petri nets.

A case study for an intelligent robotic system is conducted, a Coordination structure with one dispatcher controlling three coordinators is built to model a proposed robotic assembly system. The system is implemented using TokenPasser, and the results are analyzed to allow judgment of the tool.

CHAPTER 1
INTRODUCTION

1.1 Motivation

1.1.1 System Specification

Perhaps the most critical step in system development is the specification stage. It has been estimated that the cost of reworking an error discovered in the coding phase of a project is 50 to 200 times more expensive than fixing an error discovered during specification [4]. In addition to reducing the cost (and development time) of a system, a good specification insures that the program does the desired task, and allows better estimates of the functionality and the time to produce a system.

Some products allow automatic translations of specifications into runnable code. These tools help with system design in many ways, including

- Reducing the number of errors introduced in converting specifications to code.
- Allow quicker adaptation to changed specifications.
- Allow easier testing of the results of specification modifications.
- Assisting the developers in program verification.

1.1.2 Distributed Hierarchical Systems

Hierarchical distributed systems have been becoming more common, and more important. Hierarchical controllers process commands sent from higher levels, synchronize activities of lower level machines, and report states back to other people or processes. If this controller is controlling a distributed system, most processes will be asynchronous, synchronizing only when the controller requires it. A well designed

hierarchical controller will also allow modular connections of controlled processes, allowing for a much more versatile system.

1.2 Petri Nets for Analysis of Distributed Systems

Due to their ability to allow both synchronous and asynchronous activities, Petri nets are good tools for simulating and modeling distributed systems. Petri nets also allow rigorous mathematical analysis, including guaranteeing liveness, deadlock properties, reversability and boundedness. Since stochastic Petri nets can also be modeled as Markov chains, models can be mathematically analyzed to allow performance estimates. Finally, conventional Petri net graphical notation is intuitively easy to understand, thus allowing people to easily (if informally) analyze the net by playing the "token game".

Since Petri nets have nice specification properties (easy to understand, rigorously defined, versatile, analyzable, diverse modeling capabilities), and they can be automatically translated to code they are an ideal tool for rapid prototyping of distributed systems.

1.3 Goals of This Project

Given the above motivation, the TokenPasser code was written to allow a user to define colored Petri net transducers (CPNT) which pass tokens across internet domain ethernets to allow rapid prototyping of distributed hierarchical control systems. TokenPasser, when compiled with a suitable net definition file, allows the user to run any number of CPNTs which can communicate with the controller CPNT via sockets. The user can also enable a display of any or all of the CPNTs to allow monitoring of the progress of commands in the net. With the display off, the program runs at full speed, allowing testing of speed and system loading of the designed net. Finally, routines or programs can be attached to each transitions in the net,

allowing more complete prototyping of the system, by attaching the routines which will actually be in the final system, or stubs which simulate the output or delay of the expected routine.

1.4 Application: The CIRSSE Platform

A platform system for robotic construction in space is currently under development at the NASA Center for Intelligent Robotic Systems for Space Exploration, Rensselaer Polytechnic Institute. The system consists of a platform with two PUMA manipulators mounted on moving bases. A vision system with five cameras is incorporated into the system for object identification and location determination. The task scenario for the system is to assemble strut structures in a dynamic and uncertain environment on space stations.

A major difficulty faced by the system is the long delay in communication between earth and the space station, therefore in order to make the system able to execute the construction in space, an intelligent control system with minimum human interaction has to be designed for the system.

To this end, the theory of **Hierarchical Intelligent Control System** developed by Saridis and his colleagues [20, 21, 24] has been applied here to design the system architecture of the platform system. This is accomplished by arranging the system into three levels: the Organization, Coordination, and Execution Levels, hierarchically ordered according to the principle of *Increasing Precision with Decreasing Intelligence*. The function of the Organization Level is to define construction missions and generate the high level task plans for some specific assembly tasks. The plans include the specification of structure configuration, strut/node assembling sequence, and motion commands. The Coordination Level serves as an interface between the Organization and Execution levels. Its main function is to

translate the higher level task plans into the specific operation instructions and coordinate their execution. Finally, the Execution Level is to execute the instructions using devices in this level.

The specification and testing of the coordination level of the platform system is an ideal test for the TokenPasser program, and the case study section of this paper deals with the development and testing of that communications protocol.

1.5 Thesis Outline

This thesis is organized in five chapters and two appendices. Chapter 2, the literature review, is divided into two sections. The first section discusses Petri nets, Petri net languages, and Petri net transducers. The second section discusses some of the Petri net tools available for system modeling and creation, then goes into describes in further detail two tools for distributed system design. Chapter 3 presents the problem statement in terms of a project specification, then details the design of the project. Chapter 4 presents a case study, the dispatcher for a distributed robotic arm control system. The chapter presents the physical system being used, details the design of the Petri net transducer controllers for each subsystem, and provides a structural analysis and the results from a distributed simulation of this system. Chapter 5 summarizes the thesis with conclusions and gives suggestions for further development of the TokenPasser tool. Appendix A gives the C code for the TokenPasser program. Appendix B gives the C code for the descriptions of the developed Petri nets.

CHAPTER 2
LITERATURE REVIEW

2.1 Petri Net Theory

2.1.1 Petri nets

In this section we give a brief introduction to Petri net theory. Note that although only ordinary Petri nets are treated here, we will use the concept of colored Petri nets in our model for the coordination level, since it has been proven that as long as the number of colors is finite the colored Petri net model is equivalent to a one or more ordinary Petri nets[18].

Petri nets are tools for modeling the dynamic behavior of discrete event systems. They consist mainly of two types of elements: places and transitions. The places represent the state of the system, while the transitions represent events which change the state of the system. A place can contain a non-negative number of tokens. The state of the system modeled by a Petri net is given by its marking (the number of tokens in each place). The system evolves by firing transitions according to the execution rule described in definition 2.4.

Definition 2.1 A *Petri net* is a quadruple,

$$N=(P,T,I,O) \text{ where:}$$

1. P and T are finite sets of *places* and *transitions*, where

$$P \cap T = \emptyset \text{ and } P \cup T \neq \emptyset$$

2. $I: P \times T \rightarrow Z$ is the *input function.*

3. $O: P \times T \rightarrow Z$ is the *output function.*

where Z is the set of natural numbers.

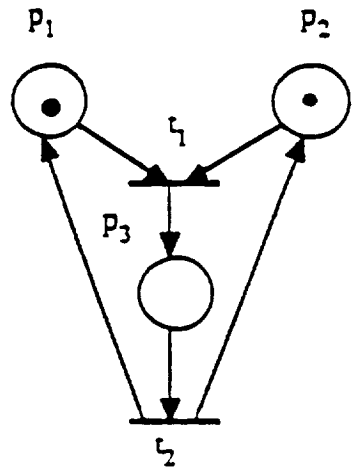

Figure 2.1: A Petri net with tokens

A Petri net can be represented by a bipartite directed multigraph, the *Petri net graph.* Places are represented by circles, and transitions by bars. There is an arc joining a place p to a transition t iff $I(p,t)\neq 0$, then p is called the input place of t. Analogously, there is an arc from t to p iff $O(p,t) \neq 0$, then p is called the output place of t. Natural numbers I(p,t) and O(p,t) are called the weights of the arcs.

Definition 2.2 A *marking* m of a Petri net N is a function $m:P\rightarrow Z$. m gives the number of tokens in each place $p \in P$.

A token can be represented by a dot. Figure 2.1 shows a Petri net with it's initial marking $m_0(p_1) = m_0(p_2) = 1$, $m_0(p_3)= 0$.

Definition 2.3 A transition t is *enabled* with respect to a marking m iff $m \geq I(t)$

Definition 2.4 (execution rule) *Firing* an enabled transition t consists of removing $I(p,t)$ tokens from each input place p and adding $O(p,t)$ tokens to each output place p. Let m_1 be the new marking resulting from firing t under the marking m_0, then $m_1 = m_0 + O(t) - I(t)$.

Definition 2.5 (reachability set) The reachability set, R(m), for a Petri net N with initial marking m is the set of all markings of N which can be reached from m by firing a finite number of transitions of N.

The following are important properties for Petri nets:

Definition 2.6 (Deadlock) A *deadlock* occurs in a Petri net when a marking is reached where no transitions in the net can be fired from that point on.

Definition 2.7 (Liveness) A Petri net is *live* with respect to a marking m if, for any marking in R(m), it is possible to fire any transition in the net either immediately, or after firing a sequence of other transitions. Liveness guarantees the absence of deadlocks.

Definition 2.8 (Reversibility) A Petri net is *reversible* with respect to a marking m if for every m'$\in$ R(m), m$\in$R(m'). Reversibility guarantees that the system modeled by the Petri net can re-initialize itself. This is important for error recovery.

Definition 2.9 (Boundedness) A Petri net is *bounded* with respect to a marking m if there exists a finite number k, such that, for any marking in R(m) the number of tokens in each place in the Petri net is less than or equal to k. A net which is bounded by the value k is said to be k-bounded, if k = 1, the net is "safe".

2.1.1.1 Petri Net Languages

The purpose of introducing Petri net language is to characterize the behavior of a Petri net model by the specification of action sequences of the net. Although various formulations for PNL have been suggested in the literature [19], in order to be consistent with our definition of Petri net transducer which will be introduced later, we give a general definition for PNL.

Definition 2.10 A *Petri net language* generated by a *labeled Petri net* $\gamma = (N, \Sigma, \beta, \mu, F$ is a set of strings over Σ

$$L(\gamma) = \beta(\alpha) \in \Sigma^* \| \alpha \in L(N, \mu) \text{ and } \delta(\mu, \alpha) \in F$$

where

- N = (P,T,I.O) is a *Petri net* with the initial marking μ.
- Σ is a finite *alphabet.*
- β :T$\rightarrow$ $(\Sigma \cup \{\lambda\})$ is a *labeling function.*
- F $\subseteq$ R(μ) is the set of *final markings.*

Note that the superscript * denotes the set of all strings of symbols formed from the stared alphabet, including the empty string.

Different types of PNL can be obtained by considering various restrictions placed on the labeling function β and the final marking set F.

2.1.2 Petri Net Transducers

Definition 2.11 A *Petri net transducer* (PNT), M is a 6-tuple,

$$\text{M} = (\text{N}, \Sigma, \Delta, \sigma, \mu, \text{F}) \text{ where}$$

- N=(P,T,I.O) is a *Petri net* with an initial marking μ.
- Σ is a finite *input alphabet.*
- Δ is a finite *output alphabet.*
- σ is a translation mapping from T$\times(\Sigma \cup \lambda)$ to finite sets of Δ^*.
- F$\subseteq$R(μ) is a set of final markings.

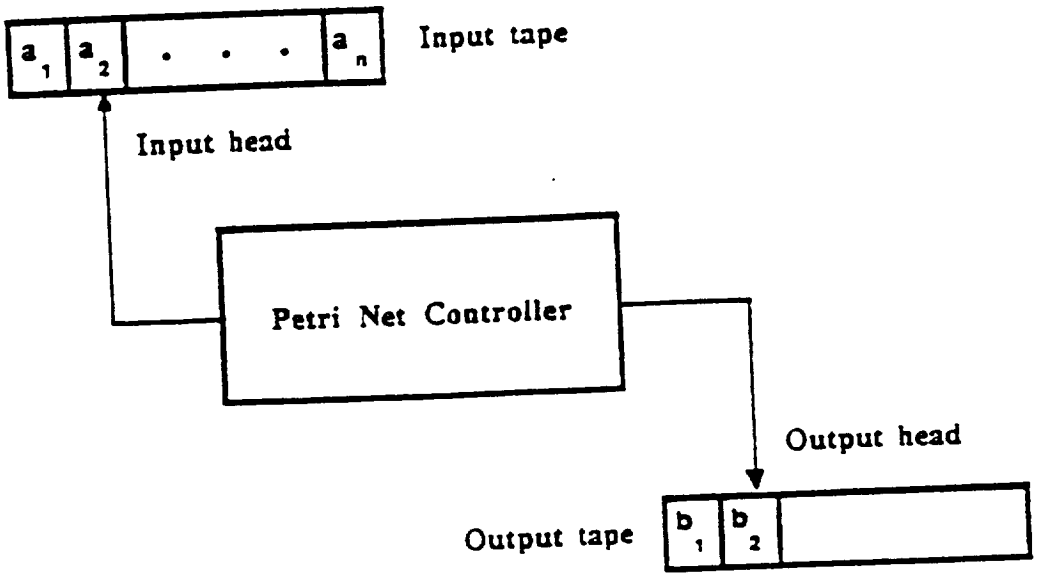

Figure 2.2: The configuration of PNTs

Physically, Σ may represent the set of primitive tasks for task planning, and Δ the primitive set of operations for task execution.

A PNT can be pictured as shown in figure 2.2. There are three parts to a PNT: an *input tape*, a *Petri net controller*, and an *output tape*. The behavior of a PNT can be conveniently described in terms of the configuration of the PNT. A *configuration* of PNT M is defined as the triple (m, x, y) where $m \in R(\mu)$ is the current state (or marking) of Petri net M; $x \in \Sigma^*$ is the input sting of the remaining input tape, with the leftmost symbol of x under the input head; $y \in \Delta^*$ is the output string emitted up to this point.

An investigation of the language properties of PNTs in [25] indicates that PNTs are self-consistent models, and therefore can be used to model that dispatcher and coordinators consistently. Wang [25] discusses the advantages of synchronous composition as a mechanism to coordinate two PNTs, and the language resulting from this composition.

2.1.3 Coordination Structures

A coordination structure (CS) is constructed in [23] by integrating Petri net transducer models of the dispatcher and coordinators through a set of connection points. The connection constraints in the definition of a CS guarenetee that each coordinator only receives tasks from the dispatcher when the coordinator is available, and the coordinator can only report the execution results when the dispatcher is ready.

Two useful theorems are proven in [23]. They are:

Theorem 1: The Petri net underlying a CS is bounded if the Dispatcher and all of the Coordinators are bounded.

Theorem 2: The Petri net underlying a CS is live if the Dispatcher and all the Coordinators are live.

2.2 Current Tools

2.2.1 Petri Net Tools

Petri nets have been extensively used for modeling distributed systems, They are very popular because of their capability of clearly describing concurrency, conflicts and synchronization of processes.

There are many Petri net tools which have been developed to assist in program design. These tools generally fit into one of two classes:

1. Design tools which assist in the construction of Petri nets, and allow one to determine several properties of the nets via either simulation or mathematical analysis[11, 15, 13].

2. Tools which assist in the creation of logic controllers. Normally these tools take a Petri net description of a system and automatically translate that description into some sort of executable code[16, 2, 8, 17].

Realistically the second group of tools is only likely to be used on nets which have been verified (for whatever criterion the designer is interested in) with one or more of the first set of tools.

This is a brief summary of the leading Petri net tools:

2.2.1.1 GreatSPN

This tool, as described in [7], provides a graphic editor for easy construction and editing of Petri nets with timed, immediate, and stochastic transitions. It is currently being expanded to include colored nets. It's analysis capabilities include calculation of place and transition invariants, boundedness, steady state token distribution, and time based performance. The performance can be calculated through either "Monte Carlo" simulations, or through analysis of the embedded Markov chain. GreatSPN also allows the user to debug the system by playing the "token game".

GreatSPN is somewhat limited in it's ability to analyze large nets, and also doesn't support connection of any two previously designed nets.

2.2.1.2 SPNP

This tool, as described in [10] reads a text description of a Generalized Timed Petri Net model, builds the embedded Markov chain, and the reachability graph for that model, solves the steady state equations for the Markov chain, and computes any "resource estimates" (means and distributions) requested in the model input.

This analysis is comparatively fast, and can handle very large nets, however writing the text description of any large net is somewhat difficult, and is likely to be error prone. To simplify writing the text description, there do exist tools for translating GreatSPN net descriptions into the input files for SPNP.

2.2.1.3 Design/IDEF, Design/CPN

Design/IDEF[11] translates from an IDEF specification to a Colored Petri Net description. These descriptions can then be analyzed by the Design/CPN package by playing the "token game". The Design/CPN package also supports creation and editing of Colored Petri Nets, including the ability to combine two different nets by substitution, invocation, or fusion. The CPN package also supports attaching code segments to transitions, so that these code segments can calculate values for output tokens, or perform other functions.

2.2.1.4 GRASPIN

GRASPIN[11] supports a graphic design of Predicate/transition nets and Pr/E nets. The tool can check the consistency, completeness and termination properties of the nets. The tool allows simulation of these nets, reachability and liveness analysis, transformation from P/T nets to specifications, and finally compilation of the net to lisp functions.

2.2.1.5 PACE

Pace [11] system allows graphical editing of hierarchical nets, with inhibitor arcs (an extension which makes Petri nets equivalent to Turing machines, but reduces the analyzableity of the net). From this description the tool can generate a C program with equivalent behavior.

The analysis available with this tool is a "token game" simulator with both forward and backward simulation, breakpoints, and modification of the net during simulation.

2.2.1.6 TEBE

TEBE [11] takes the 1-safe net and produces a reduced net performing the same function.

2.2.2 Distributed System Tools

Other than Petri net modeling systems, which could model distributed systems by modeling the the communication network, none of the Petri net tools found dealt with distributed systems. There are, however, several systems which allow design of distributed controllers. Two approaches are detailed here. The first is interesting due to the improvement in the development times of the built systems. The second is an example of a distributed system controller design tool built on the Unix system.

2.2.2.1 C-nets

Murata et al. [16] Describe a Control net (C-net), an enhancement of a Petri net which they use to build a micro-computer based controller which controls some machinery by running the C-net.

A C-net is defined by the 10-tuple CN = (P,T, I, O, δ, φ, η, U, V, M) where P, T, I, O and m are the same as in the standard Petri net model. δ, φ, and η are called process I/O functions, and U and V are process status functions. These functions are used to define process interfaces and process statuses.

Process I/O Functions Let A be a set of control signals (x_i) and E a set of observable signals (y_{ij}) then δ:P$\rightarrow$A, φ:P$\rightarrow$E, and η:T$\rightarrow$E are defined as follows:

$$\delta(p_i) = x_i, \qquad (x_i \in A, p_i \in P)$$

$$\varphi(p_i) = (y_{i1}, yi2, \cdots, y_{in}). \qquad (y_{ij} \in E, p_i \in P)$$

$$\eta(t_i) = y_{ik}, \qquad (y_{ik} \in E, t_i \in E).$$

When a token enters place p_i, a control signal x_i, defined by $\delta(p_i)$ is put out and a machine action is triggered. The token stays in the box until one of the input signals $y_{i1}, \cdots, y_{in}$ defined by $\varphi(p_i)$ is detected as a response signal of a completed action.

Process Status Functions To define the action's execution status at a place and to manage transition open/close statuses, process status functions U:P→L (L=0,1,$\cdots, q$), V:P→N (N= 0,1) are defined as follows:

$$U(p_i) = \begin{cases} 0 & \text{action associated with } p_i \text{ is executing now} \\ in & \text{action associated with } p_i \text{ is completed with return code } y_{in} \end{cases}$$

$$V(t_i) = \begin{cases} 0 & t_i \text{ is closed} \\ 1 & t_i \text{ is opened} \end{cases}$$

By introducing these functions, action execution statuses or transition operation modes can be supervised at a place or transition.

Transition Firing Rule A Transition $t_i \in$ T can be enabled at marking m_1 iff:

$V(t_i) = 1$, $M(p_i) = 1$, $U(p_i) \neq 0$, and $M_1(p_j) = 0$ for all $p_i \in I(t_i)$ and $p_j \in O(t_i)$.

By firing t_i tokens are generated in all output places and deleted from all input places. When the tokens are generated, the output signals defined at the output places of t_i are put out and U(p_i) is set at 0.

Based on this model a C-net interpreter was designed and installed on a microcomputer system. The operator could draw a C-net on a graphic display, and input control tables. The monitor displays the machine status in real time by displaying tokens in the active places.

Three experiments were described, an assembly station, a robot controller, and a general flexible manufacturing cell controller. The software development times

were reduced by 50 to 80 percent compared with the relay ladder diagram method, and the system maintenance time was reduced by 60 percent.

2.2.2.2 The Olympus Simulation System

The Olympus Modeling system[17] is an interactive, distributed model interpretation environment for bilogic precedence graphs (BPGs). BPGs, like Petri nets represent the status of the model through a distribution of tokens on nodes and edges. An interpreted BPG corresponds to a simulation model of some system.

The Olympus system is an interesting tool because it allows a user to define and simulate a distributed system with a distributed simulation.

Olympus consists of a frontend and a backend, the frontend implements the user interface, while the backend provides storage and interpretation of the model.

An Implementation Figure 2.3 illustrates an implementation of Olympus in a network of Sun workstations, using Unix processes, graphics and network protocols. The frontend is a point and select editor built on Sun's NeWS model, implemented as a NeWS client and a NeWS server. The client implements the logical aspects of the user interface, while the NeWS server process is responsible for placing images on the display.

The Olympus server (backend) is implemented as n+1 Unix processes. The first process multiplexes among the four interpretation and storage subproceses, the other n processes are used to evaluate BPG interpretations. The BPG interpretations can be defined in any language, provided that the definition can be called as a C procedure. The task interpreter uses the Sun Remote Procedure Call facility to invoke the interpretation procedure whenever the corresponding node is fired. The result of this is that the tool runs a single BPG net, with the firing of each node starting a procedure on some (possibly remote) machine. This allows the rapid prototyping and simulation of distributed systems.

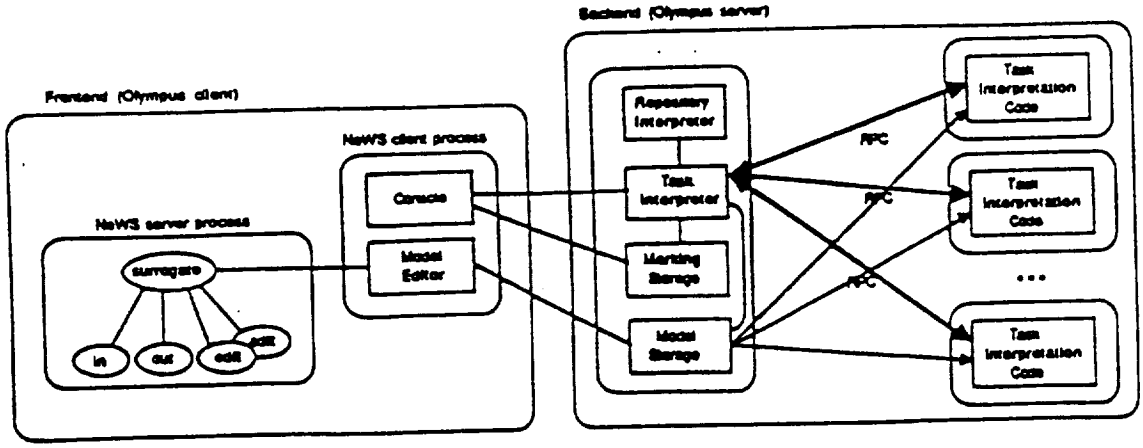

Figure 2.3: An Olympus Implementation

CHAPTER 3
PROBLEM STATEMENT

3.1 Project Specification

The object of this project is to build a tool which allows the user to define distributed hierarchy of communicating colored Petri net transducers. We wanted to be able to

- Distribute the Petri nets on different machines, with easy ways of changing the connections between different nets, or the machine on which a net runs.
- Have a graphical display, to allow the user to allow the user to inspect the progress of the net.
- Attach routines to any or all of the transitions, to allow more detailed simulations.

Because they are all common and versatile systems, this tool is being developed as a C program to run under the Unix operating system, using the X11 windowing system for any displays.

3.2 Project Design

3.2.1 Communication

The communication between the processors is an important issue. Since it was decided to build this program on the CIRSSE Unix system, the Unix methods of communication were most available.

A socket is a Unix Inter Process Communication (IPC) construct, which allows communication between processes on multiple machines. Sockets allow many options including non-blocking reading and writing, and asynchronous notification

of data arrival. Functionally sockets look like files, they can be written to, or read from with the `read(2)` or `write(2)` commands. Sockets have many communication semantics, the most convenient one was the SOCK_STREAM which provides sequenced, reliable, two-way byte streams. Sockets also use several different protocols, the most relevant one is PF_INET, which is an internet protocol, which provides enough versatility to go across ethernets.

Since all UNIX IPC is built upon sockets, sockets are the fastest built-in communication method possible, and due to the relative ease of writing and reading data using them, SOCK_STREAM, PF_INET sockets were the tool used.

3.2.2 Data Structures

To run a Petri net transducer one must be able to locate all enabled transitions. This program is also required to run a routine at each transition. Since the Petri nets are colored, and this is a Petri net transducer, each transition must have a set of tokens which enable it, and a set of "menu" commands which allow the particular transition to fire. From this we decided on a set of data structures for the implementation of the Petri net transducer: Each transition is a structure, pointing to

- the function which the transition must run,
- a list of pointers to the pre-places of the transition,
- a list of pointers to the post places.
- a list of "menu" commands which enable the transition
- a list of token "colors" which enable the transition.

```
struct transition {
    struct pre_pointer *pre_places;
    struct post_pointer *post_places;
    int (*routine) ();
    struct enabelors *enabled_by;       /*valid token "colors". */
    struct enabelors *menu_requirements;  /* this refers to the*/
                              /* values required in the "menu"*/
    int consumes_menu;        /*this increments  the menu ptr? */
    int x_loc;                /*for graphic output */
    int y_loc;                /*for graphic output */
    };
```

Each place is a pointer to a list of objects (currently integers) which are in the place. Each transition is connected to its pre and post places by a set of pointers which point to structures which point to the place. These pointers (pre_pointers, and post_pointers) also point to the routine associated with coloring each token before feeding it to a transition and decoloring it on the way out, in the case of post_pointers there is also a pointer to the routine which should be used if the data is to be sent to a remote machine.

```
struct place_ptr{
    struct place_contains *place;
    int x_loc;                /*for graphic output */
    int y_loc;                /*for graphic output */
    };

struct place_contains{
    int object;
    struct place_contains *next_contents;
    };

struct pre_pointer {
    struct place_ptr *place;
    struct pre_pointer *next_place;
    int (*decoloration_routine)();   /*returns true if it enables */
    };                               /*transition, false otherwise*/
```

```
struct post_pointer {
    int remote;                  /* =0 if local otherwise it is a   */
    struct place_ptr *place;     /* pointer to the socket if remote */
                                 /* this points to an integer with a*/
    struct post_pointer *next_place;  /* value=the position in the*/
    int (*data_trans_routine)();      /*array for the remote value*/
    int (*coloration_routine)();
    };
```

Figure 3.1 displays how all of these structures fit together.

3.2.3 Flow of Control

Each sub-net is run by one block of code. The main block sets up one socket for each of the other blocks, and prints out files giving the "address" of each socket. It then waits for the other three routines to connect to it. The auxiliary programs start up, and connect to the given socket. Each of the programs then initializes its Petri net, and starts trying to fire transitions.

Given the data structure described in the previous section it is easy to write code to run the Petri net. Specifically it looks like:

```
while (TRUE)
   {
   for (j =0; j< num_transitions;j++)
         if (transition_enabled_p(transition[j])
                  fire_transition(transition[j]);
   }
```

This code checks each transition. and if it is enabled fires it. The algorithm for determining if a transition is enabled in a colored Petri net transducer is slightly complicated, the pseudo code for it looks like this:

```
transition_enabled_p(transition){
    if (transition->menu_requirements == NULL)
        if(transition.enabled_by == NULL)
            if(at_least_one_token_in_each_preplace (transition)
                return(TRUE);
            else return(FALSE);
        else  {
            for(token=transition.enabled_by;token=token.next;
```

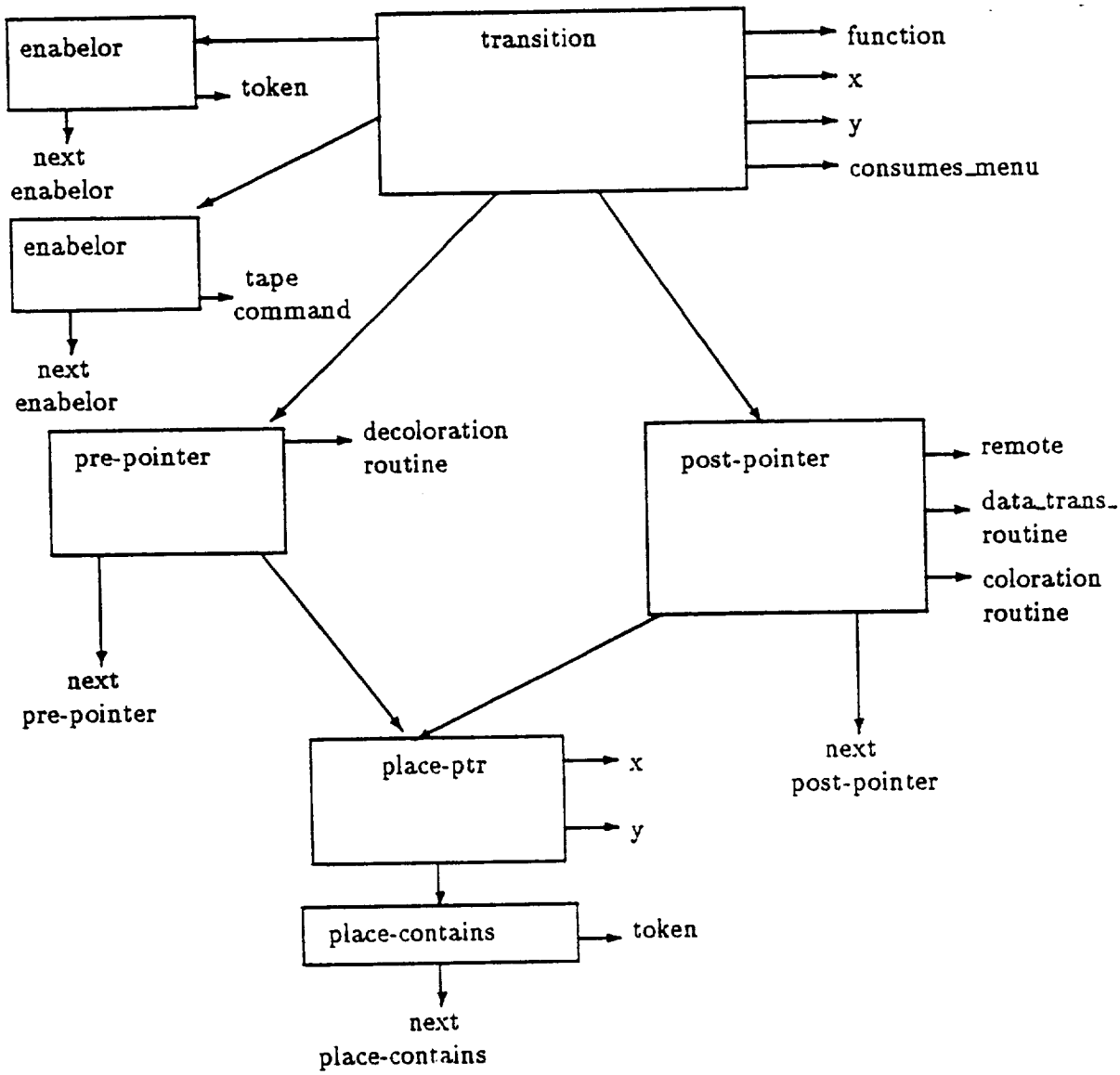

Figure 3.1: The Data Structures of the TokenPasser Program

```
                token==NULL)
              if(at_least_one_token_per_place_matches(token))
                  return (token);
              return(FALSE);  }
    else
        if(at_least_onematches(trans->menu_requirement,
                                          current_menu())
            if(transition.enabled_by == NULL)
                if(at_least_one_token_in_each_preplace (transition)
                        return(TRUE);
                  else return(FALSE);
             else  {
                  for(token=transition.enabled_by;token=token.next;
                        token==NULL)
                        if(one_token_per_place_matches(token))
                              return (token);
                  return(FALSE);  }
    return(FALSE);
}
```

Once it is determined that a transition is enabled, firing a transition consists of: removing one (appropriately colored) token from each pre-place, running the routine associated with the transition, re-coloring the token, and putting it in each post-place, and incrementing the "menu" pointer if the transition required a "menu" command to fire.

Note, one flaw with the present system is that it is impossible to specify that a certain transition requires multiple tokens from one place.

3.2.4 User Interface

The input data for the Petri net is entered in the form of a file which is compiled with each program. This program has an X-window graphical display as output. This display shows the Petri net, and the tape for the net, along with the current marking of the net. This allows the transmission of data and commands to be seen, in addition to the overall command structure for the the defined nets.

Each net is run by it's own block of code. If the user chooses to observe the

firings of the net the block of code forks to two processes, the first of which runs the net as described in the previous section. The second block monitors and updates the display, which allows the display to be moved, resized, or hidden without interfering with the progress of the firing of the net. Of course, under normal conditions, the transitions fire to quickly to allow the user to observe the net, so when the display is "on" there is a 300 mS delay built into each transition, so the user can see it flash. If the user doesn't display the net, the 300mS delay is not enabled.

CHAPTER 4
CASE STUDY

4.1 CIRSSE System Overview

4.1.1 Goals

One of the present goals of the CIRSSE platform project is to be able to assemble struts and nodes autonomously onto some well-defined structures in a space environment. For this task it has been assumed that there are only three classes of objects in the workspace of the platform: struts, nodes, and obstacles. The obstacles may travel in the workspace in some unpredictable ways, and the platform system has to avoid the possible collision with them during the process of construction.

A completely autonomous Organization Level for the platform system is not considered at the current stage. Therefore, its function is replaced by a human operator in the earth station. In order to make the Coordination Level understand the tasks from the Organization Level (which may be a human operator), a formalism, called *system command language*, for expressing the construction missions by the operators is designed, and the input to the Coordination Level will be compiled system commands (or task plans). Obviously, to reduce the communication required between the earth station and the space station, the compiler for the system command language should be hosted on the space station with the Coordination Level.

Due the constraint imposed by earth-space communication, the Coordination Level has to be able to operate under the situation that task commands from the high level arrive infrequently and erratically. To accomplish this, the dispatcher (D) of the Coordination Level should

1. Decompose the task commands into subtasks and dispatch them in order when

the relevant coordinators have signaled that they are ready for the next execution.

2. Communicate with the coordinators relatively quickly;

3. Be capable of setting up communication between any two coordinators which need information exchange.

The three coordinators of this Level are: the vision coordinator (VC), the gripper coordinator (GC) and the motion coordinator (MC). The system components of the dispatcher and the coordinators are described in sequel.

4.1.2 System Components

4.1.2.1 Dispatcher

As noted in the goals, the communication time to the dispatcher may be erratic, however, we require the dispatcher to communicate quickly with any of the coordinators. Because of this all of the coordinators are designed to allow (relatively) long waits between commands. Additionally, the dispatcher is not involved in transmission of large blocks of data. If large amounts of data is expected to be passed from one coordinator to another, then the dispatcher just instructs the coordinators to connect to each other, an allows them to communicate at whatever rate they are capable of.

The dispatcher is physically realized on Sol, a SUN4/260 workstation. The dispatcher communicates with the coordinators through THINNET, a version of ETHERNET. The communication will be implemented with IPC SOCK_STREAM type sockets, under the IF_INET protocol. This Implementation results in a minimum turn around time of $200mS$ from one processor to another and back.

4.1.2.2 Vision

The vision coordinator has the job of "looking" at an area that it is told to observe, and attempting to find something which it has been told to find there. It then gives the location of said object back to the system which asked for it, and waits for another command. The location of an object includes it's position, and orientation, and in the case of obstacles, a simple description of it's surface.

The vision system is physically realized on a VxWorks cage with a Data cube, and a Motorola MVME147 controller. The controller is connected to the ethernet, and thus to the rest of the system. The Datacube boards consist of:

SNAP The SNAP board performs non-linear transformations (comparisons and max/min determinations) in sequential digital video data.

VFIR-MKIII A video impulse response filter module. It implements a 256 arbitrary coefficient convolution. This is primarily used in edge detection, and noise filtering.

Max-SP This is capable of performing real time frame rate single point temporal and spatial filters. image merging, image subtraction and addition, and Min/Max processing.

FEATUREMAX-MKII this does advanced feature-list extraction, and histogram grams. A summation of all row or column pixels can be done in a table.

MAX-MUX This provides the MaxVideo user with software control over MAXbus data source and destination selection. This allows for easier reprogramming of the Datacube.

DIGIMAX This is a video acquisition and display module which is capable of accepting one of eight inputs. This is used to feed the information from the cameras to the ROI-STORE units.

ROI-STORE this is a frame storage module which supports user programmable video resolution and processing of regions of interest within a video image.

Since there are two DIGIMAX cards two frames can be read simultaneously. Two of the cameras are mounted on the manipulators, allowing greater diversity in the objects which can be viewed, but presenting greater challenges in calibration. Two of the remaining cameras are mounted on the ceiling of the workspace, and the remaining one has not yet been placed.

The each frame grabber is capable of reading the cameras at a rate of 30 Hz. The output of the Datacube is sent to the MVME147 which can further analyze the image and send data to other coordinators. The MVME147 does any intermediate level operations, such as Hough transforms, or line fitting.

The Uniphase laser is controlled by the Motorola MVME 135 CPU, and the MVME 340 parallel board. The laser is used to put bright points on the object, to make stereo point matching easier.

4.1.2.3 Arm

The motion system is used to move objects in the environment, and to move cameras which are attached to the robot arms. The motion system is also directly used by the vision coordinator during calibration.

The motion system is physically realized on a VxWorks cage running 5 Motorola MVME 135 boards (68020 CPUs), along with

MVME340A A parallel port board, which also supplies timer interrupts. This is used to read the sensors, and supply interrupts for the platform servo control.

VMIVME 2532A A digital I/O board, which is used primarily for switching external circuits, thus allowing software control of power to any of the manipulators.

DVME 628 A D/A converter, for supplying motor currents to the arms. The digital signal is converted to analog, and then run through a servo amplifier and fed to the joints in the system.

MVME 224-1 Four MBytes of shared memory.

XVME 556 A 16 channel A/D converter, which is currently unused, but may be used for reading encoders.

Whetdco Encoder A VME 3570 Optical shaft encoder, used for reading the position of the carts on the Aronson platform.

VME 7016 A VME Q Bus controller, used to control the Puma.

332 XT Eight channel serial interface. This will be used to control the gripper.

The 68020s connect via the databus to a D/A board which feeds currents to a Puma 560,and a puma 600 arm. The puma arms have absolute position potentiometers, and torque sensors at all of their joints. Each arm is mounted on an Aronson platform which gives each arm three more degrees of freedom. The arms and platform are controlled by Kali, [22] an integrated path planner/arm controller.

4.1.2.4 Gripper

The gripper will be used to actually grasp struts and nodes. It must be able to sense when there is an object between its "fingers", report the position that the fingers are in, and the force they are applying. The gripper is a pneumatically controlled gripper. Each gripper is equipped with an crossfire sensors, and force sensors, and is mounted on a Lord force/torque sensor, which is mounted on the end of the puma arm. The gripper controller is a Motorola 68HC11 based controller, which communicates with the VxWorks cage through the 332 XT serial interface in the VxWorks cage.

4.2 System Command Language and Task Grammar

As mentioned in the previous section, a system command language is necessary for the Coordination Level to interact with the human operators in the Organization Level. This high level language will also make the programming of the construction missions much easier for operators.

A general formalism for the system command language can be defined by following the syntaxes of the existing high level languages, such as Pascal and VAL-II. One of such examples is the command formalism developed by Noreils and Chatila (1989) for a mobile robot system. However, since this paper is concentrated on the Coordination Level of the platform system and the inputs to that Level are only the compiled system commands (that is, sequences of tasks which are directly related to the operation of the Level), no attempt to specify a formal language for system commands will be made here. Instead, we define the following *task grammar* G to represent the compiled system commands to the Coordination Level:

$$G = (S, N, \Sigma_o, P)$$

where

$$\begin{aligned}
\Sigma_o &= \{calR, calV, move, slave, approach, release, grasp, \\
&\quad findS/N, findOBScontinue_vision\} \\
N &= \{S, V, M, M_s, M_v, H_s, H_{s1}, H_{s2}, H_v, H_{v1}, H_{v2}\} \\
P &= \{S \rightarrow calR\ M \\
&\quad M \rightarrow calV\ V \mid move\ M_s \\
&\quad M_s \rightarrow move\ M_s \mid approach\ M_s \mid approach\ H_s \\
&\quad H_s \rightarrow release\ H_{s1} \\
&\quad H_{s1} \rightarrow grasp\ H_{s2} \mid move\ H_{s1} \mid approach\ H_{s1} \mid calV\ V \\
&\quad H_{s2} \rightarrow release\ H_v \mid release\ M_s
\end{aligned}$$

$$V \rightarrow findS/N\ V \mid findOBS\ V \mid move\ M_v \mid approach\ M_v \mid slave\ V_t$$

$$V_t \rightarrow continue\_vision\ V$$

$$M_v \rightarrow move\ M_v \mid approach\ M_v \mid approach\ H_v \mid V$$

$$H_v \rightarrow release\ H_{v1}$$

$$H_{v1} \rightarrow grasp\ H_{v2} \mid move\ H_{v1} \mid approach\ H_{v1} \mid V,$$

$$H_{v2} \rightarrow release\ H_v \mid release\ M_v$$

$$V, M, M_s, M_v, H_s, H_{s1}, V_t, H_v, H_{v1} \rightarrow S\}$$

Σ_o represents the set of task primitives (terminal symbols), N is the set of non-terminal symbols with S as the start symbol, and P the set of production rules for deriving task plans (or language).

The task grammar G characterizes the basic task precedences in the operation of the platform system. With this given grammar, the problem of establishing the Coordination Level becomes that of constructing a coordination structure which is capable of processing all the task plans generated by G. This will be accomplished in th following section by giving the individual Petri net transducers for the dispatcher and coordinators.

4.3 Coordination Structures

4.3.1 The Petri Net Transducer for the Dispatcher

The simplified Petri net model for the dispatcher, shown in figure 4.1 consists of 12 places and 16 transitions. A transition generally represents dispatching a command to perform a specific task, while a place represents the state of the system. These places and transitions are specified as follows:

Transitions:

CalR: Calibrate the Robot arm.

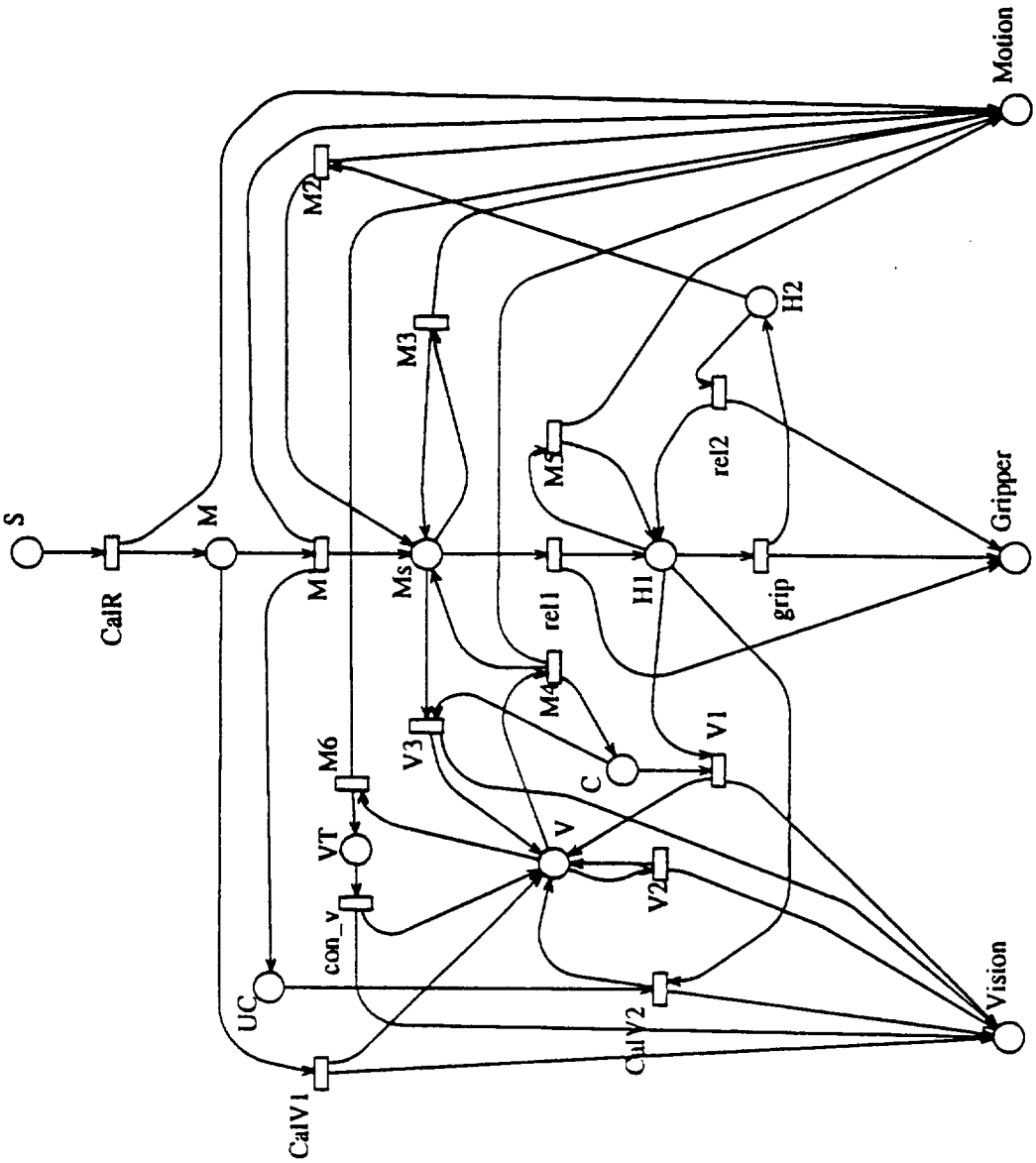

Figure 4.1: The Simplified Dispatcher Architecture

M1, M2, M3, M4, M5, M6: Send a Move or Approach command to the Motion Coordinator.

Rel1, Rel2: Send a release command to the Gripper Coordinator.

Grip: Send a grip command to the Gripper Coordinator.

V1, V2, V3: Send a find obstacle. find strut, or find node command to the Vision Coordinator.

CalV1, CalV2: Send a Calibrate command to the Vision Coordinator.

Con_V: Continue the interrupted vision task.

Places:

C: Holds a token if the vision system has been calibrated.

UC: Holds a token if the vision system is not calibrated, and at least one move command has been done.

S, M, Ms, H1, H2, V Vt: These places correspond to the non-terminals in the grammar.

Vision, Gripper, Motion: These three places each represent four input and output places and semaphores for each of the three subnets.

The full Petri net model for the dispatcher, 4.2 consists of 37 places and 32 transitions. For simplicity. the simplified model will be discussed, as understanding all of the places and transitions is not necessary for understanding the net.

The input alphabet for the dispatcher is the set of primitive tasks from the organization level. The output alphabet . i.e., the set of primitive control actions in the Coordination level is $\Delta_d = \Delta_0 = \Sigma_v \bigcup \Sigma_m \bigcup \Sigma_g$, where Σ_v, Σ_g, and Σ_m are the input alphabets for the vision coordinator. the motion coordinator, and the gripper coordinator. We have :

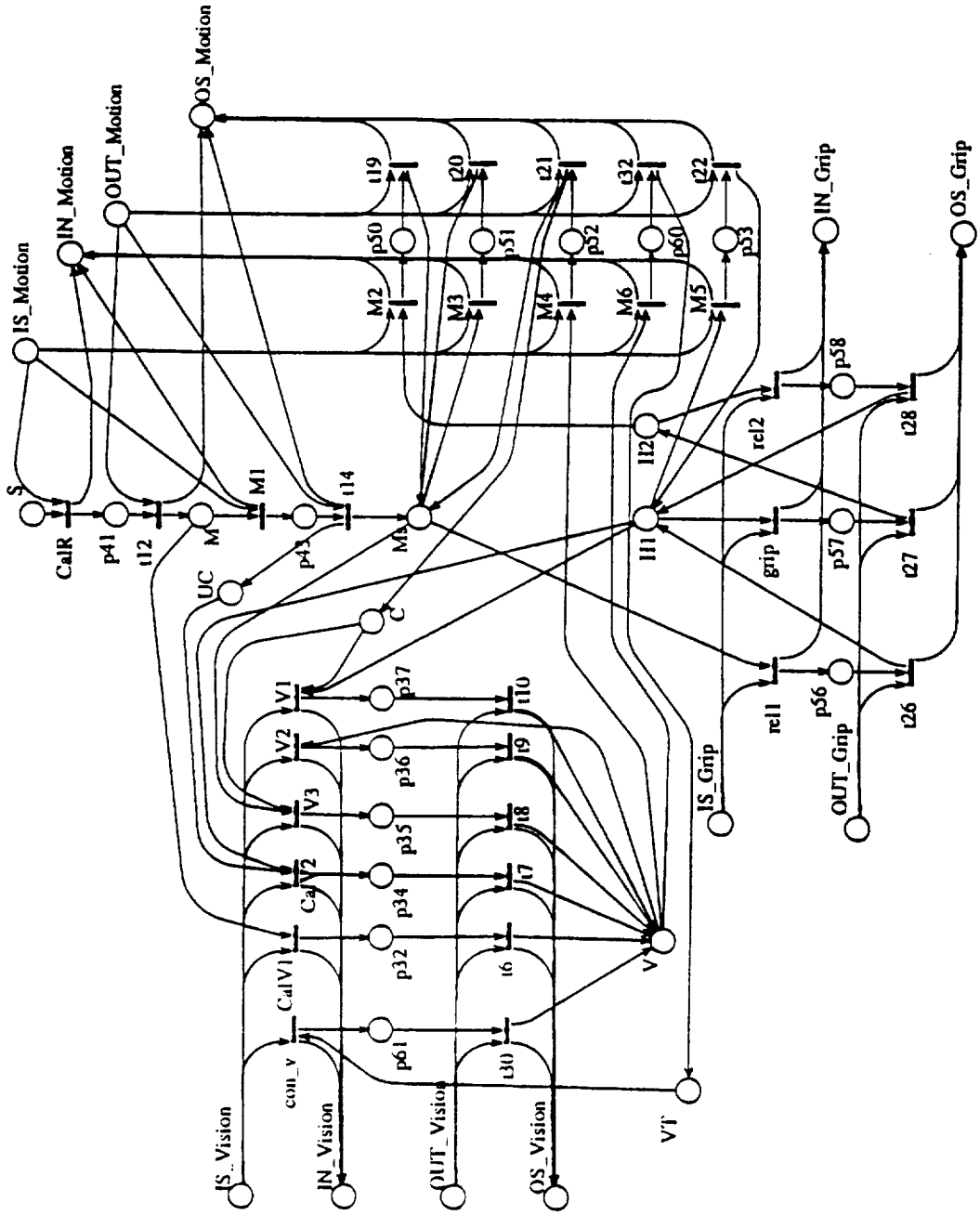

Figure 4.2: The Dispatcher Architecture

$$\Sigma_d = \Sigma_0 = \{Calibrate_v,\ Look_v,\ Find_v,\ Slave,\ Continue\_vision,\ Calibrate_m,$$
$$Move_m,\ Approach_m,\ Grasp_g,\ Release_g\}$$

The translations for all of the σ_ds for the dispatcher is as follows:

The translation mapping for $\sigma_d(Con\_V.\ continue\_vision)$ can be specified as:

```
send_tape(vision_socket, Insert_tape_immediate, Continue);
```

The translation mapping for $\sigma_d(V_n,\ Look)$ can be specified as:

```
send_tape(vision_socket, Append_to_end, Look, Return);
```

The translation mapping for $\sigma_d(V_n,\ Find_v)$ can be specified as:

```
send_tape(vision_socket, Append_to_end, Find, Return);
```

The translation mappings for $\sigma_d(Rel1,\ Release_g)$ and $\sigma_d(Rel2,\ Release_g)$ can be specified as:

```
send_tape(gripper_socket, Append_to_end, GoPosition, Return);
```

The translation mapping for $\sigma_d(Grip,\ Grasp_g)$ can be specified as:

```
send_tape(gripper_socket, Append_to_end, GoPosition, Return);
```

The translation mappings for $\sigma_d(CalV1.\ Calibrate_v)$ and $\sigma_d(CalV2,\ Calibrate_v)$ can be specified as:

```
send_tape(vision_socket, Append_to_end, CalV, CalV, CalV, CalV,
          CalV, CalV, CalV, Return);
```

The translation mapping for $\sigma_d(CalR.Calibrate_m)$ can be specified as:

```
send_tape(arm_socket, Append_to_end, CalR);
      /* we are considering removing this command from the */
      /* grammar, as it only needs to be implemented every */
      /* 6 months, not every time the system is brought up */
```

The translation mapping for $\sigma_d(M_n, Approach_m)$ can be specified as:

```
send_tape(arm_socket, Append_to_end, Approach);
```

The translation mappings for $\sigma_d(M_n, Move)$ and $\sigma_d(M_n, Slave)$ can be specified as:

```
send_tape(arm_socket, Append_to_end, Move);
```

All other transitions of the dispatcher are internal operations.

4.3.2 Petri Net Transducer for the Vision Coordinator

The Petri net model of the vision coordinator in figure 4.3 consists of 15 places and 16 transitions. The transitions generally represent lower level routines, while the places represent the state of the system. The places and transitions are specified as follows:

Transitions:

Look: is fired to direct the vision system to "look" at a given location.

NMv: No Move is fired if no arm movement is required to "look" at a given location.

Mv: Move is fired if arm movement is required to "look" at a location.

BMv: Begin Move is fired to send the move command to the motion coordinator.

Cal: Calibrate is fired when the calibrate command is given.

Find: is fired to give the command to find a strut, node or obstacle in the scene that the vision system is already looking at.

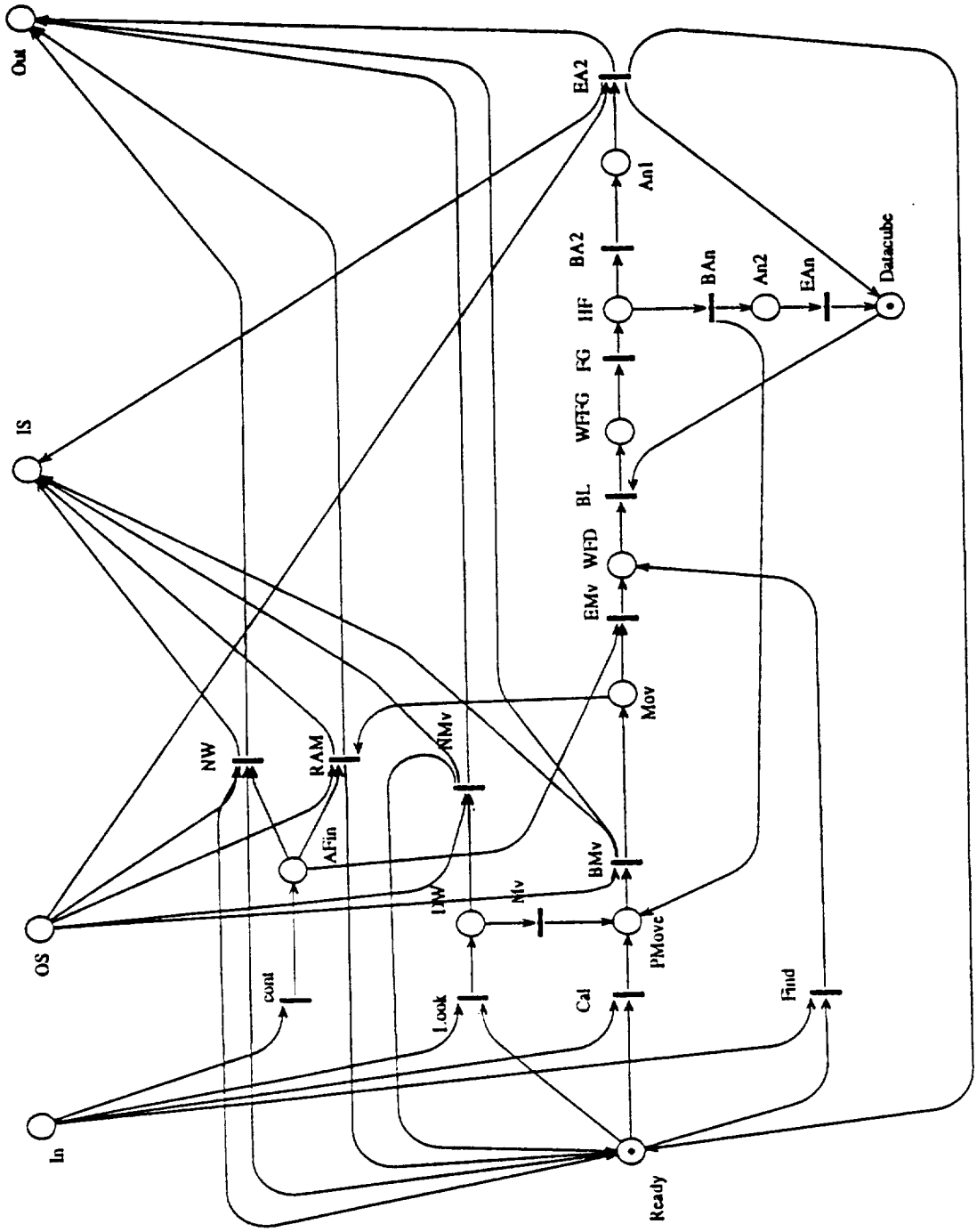

Figure 4.3: The Vision Petri Net Transducer

EMv: End Move is fired when the Motion system returns a non-error value and the menu command is calibrate.

BL:Blocks until the Datacube is free.

FG: the Frame Grabber gets the picture from the cameras, and sends the data to the Datacube.

BAn: Begins Analysis of the picture in the case when another move command must be sent to the arms.

EAn: Analysis is complete, and the Datacube is returned to the available state.

BA2: Analysis Begins for the case when the task is about to be completed.

EA2: Analysis Ends, and the state of the net is reset to ready, while the results are sent to Out.

Cont: Fired when a "continue action" command is given. Informs the Vision net that the event it was waiting on (an arm motion) is completed.

NW: Fired when the net receives a "continue" command when it isn't waiting.

Places:

In: The input place.

Out: The output place

Ready: Marks the availability of the system.

IS, OS: The input and output semaphores.

Datacube: Marks the availability of the Datacube.

An1, An2: Analyzing: the Datacube is processing information.

AFin: The arm is finished with it's action, or a continue command was sent.

Mov: The arm is moving.

PMove: Prepare to Move the arm

WFD: Wait For Datacube.

HF: Have Frame.

DW: Decide Where cameras need to be to "look" at the given location.

WFFG: Wait For Frame Grab.

The subtasks to be processed by the vision coordinator are: $\Sigma_v = \{look, Find, calibrate\_vision, return, continue\}$. The output alphabet Δ_v consists of hardware related operations for the camera devices, and commands to the Motion coordinator.

The translation mapping σ_v for the vision coordinator is expressed as follows: The translation mapping $\sigma_v(Look, look)$ can be specified as:

```
look_at(x,y,y,cameras_allowed)
    float x,y,z;              /*The location we want to look at*/
    boolean cameras_allowed[]; /*specifies cameras we may use*/
    {
    int i;
    for(i=0; i< NUM_CAMERAS; i++)
          cameras_allowed[i] &= can_be_pointed_at(x,y,z, i);
    if (number_of_allowed(cameras_allowed) < 2) return (ERROR);
    else enable_two_best_cameras(cameras_allowed);
    return(enabled_camera_descriptor);
    }
```

The translation mapping $\sigma_v(Mv, look)$ can be specified as:

```
calculate_desired_arm_position(x,y,y,cameras_enabled)
    float x,y,z;                The location we want to look at*/
    boolean cameras_enabled[];  /*specifies cameras we may use*/
    {
    if(number_of_cameras_mobile(cameras_enabled) != 1)
          return(ERROR);
    else load (where_the_camera_should_be(x,y,z),
        DESIRED_ARM_POSITION_TABLE);
    return(OK);
    }
```

The translation mapping $\sigma_v(Cal,$ *calibrate_vision*$)$ can be specified as:

```
load_desired_arm_positions_for_calibration();
analyze = &analyze_calibration_card_routine;
```

The translation mapping $\sigma_v(Find, Find)$ can be specified as:

```
setup_analyze_routine(tape_cmnd, high_precision)
boolean high_precision;
int tape_cmnd;
{
if(high_precision){
    switch(tape_cmnd) of
       case  strut:    analyze_routine = &find_strut_routine_hp;
       case  node :    analyze_routine = &find_node_routine_hp;
       case  obstacle: analyze_routine=&find_obstacle_routine_hp;
       }
    }
else
    switch(tape_cmnd) of
       case  strut:    analyze_routine = &find_strut_routine;
       case  node :    analyze_routine = &find_node_routine;
       case  obstacle: analyze_routine = &find_obstacle_routine;
       }
    }
}
```

The translation mappings $\sigma_v(BMv,$ *calibrate_vision*$)$ and $\sigma_v(BMv,$ *look*$)$ can be specified as:

```
read_desired_arm_position();
send_tape(dispatcher_socket,Insert_tape_immediate,Slave,Continue);
```

The translation mappings $\sigma_v(FG,$ *calibrate_vision*$)$ and $\sigma_v(FG, Find)$ can be specified as:

```
grab_frames(cameras_enabled);
```

All other transitions are internal operations.

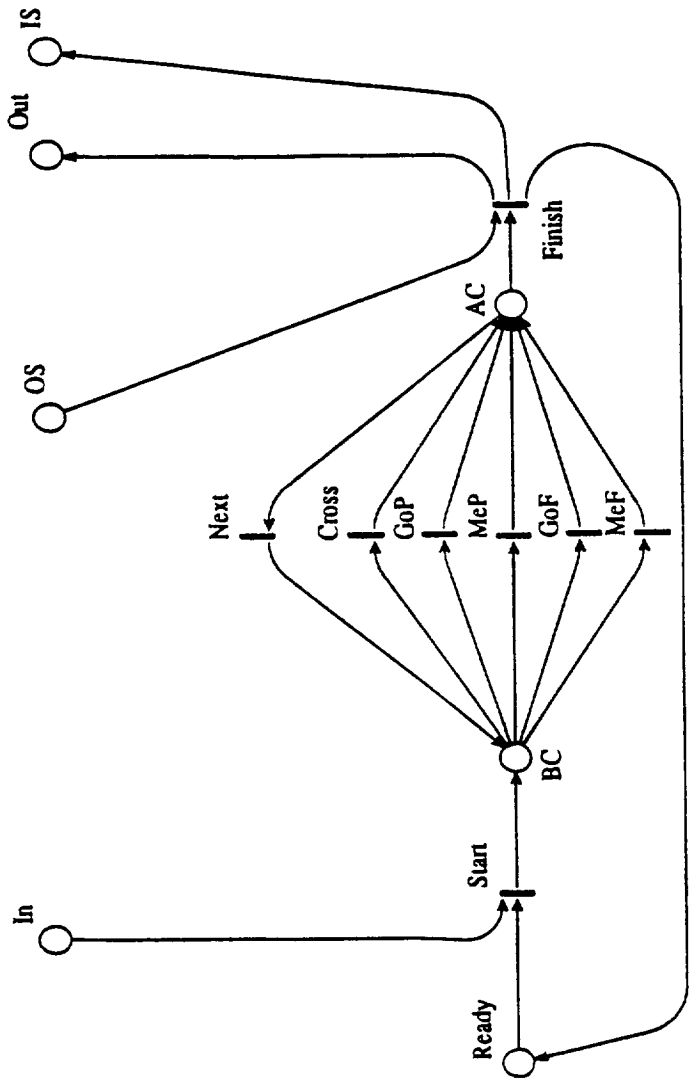

Figure 4.4: The Gripper Petri Net Transducer

4.3.3 Petri Net Transducer for the Gripper Coordinator

The Petri net model of the gripper coordinator in figure 4.4 consists of 7 places and 7 transitions. The transitions generally represent lower level routines, while the places represent the state of the system. The places and transitions are specified as follows:

Transitions:

Start: Checks that the gripper is ready to run the next command.

Cross: Checks the crossfire sensor, returning true or false.

GoP: Go Place closes the gripper to a desired width.

MeP: Measure Place measures the size of the gripper opening.

GoF: Go Force closes the gripper until the desired force is reached.

MeF: Measure Force returns the force that the gripper is applying

Next: Sends the token back if there is another command in the menu.

Finish: Returns the token to wait for another command.

Places:

In: Input place.

Out: The Output place.

IS, OS: The input and output semaphores.

Ready: Gripper Ready for the next command.

BC: Before Command.

AC: After Command.

The subtasks to be processed by the gripper coordinator are $\Sigma_g = \{crossfire, GoToPosition, GoToForce, MeasurePosition, MeasureForce\}$. The output alphabet consists of hardware related operations for the gripper.

The translation mappings for σ_g the are:

The translation mappings $\sigma_g(Cross, crossfire)$ can be specified as:

```
crossfire(){
if (data_from_crossfire_sensors == BLOCKED) return (TRUE);
else return(FALSE);
}
```

The translation mappings $\sigma_g(GoP, GoToPosition)$ can be specified as:

```
goto_position(desired_position){
    write_val = calculate_controller_value(desired_position);
    write_to_controller(write_val);
    if(whatever_the_error_conditions_are) return(ERROR);
    else return (OK);
}
```

The translation mappings $\sigma_g(GoF, GoToForce)$ can be specified as:

```
goto_force(desired_force){
    write_val = calculate_controller_value(desired_force);
    write_to_controller(write_val);
    if(whatever_the_error_conditions_are) return(ERROR);
    else return (OK);
}
```

The translation mappings $\sigma_g(MeP, MeasurePosition)$ can be specified as:

```
measure_position();
```

The translation mappings $\sigma_g(MeF, MeasureForce)$ can be specified as:

```
measure_force();
```

All other transitions are internal operations.

4.3.4 The Petri Net transducer for the Arm Coordinator

The Petri net model of the motion coordinator in figure 4.5 consists of 13 places and 9 transitions. The transitions generally represent lower level routines, while the places represent the state of the system. The places and transitions are specified as follows:

Transitions:

Start:Checks that the User Program is ready to submit the next command.

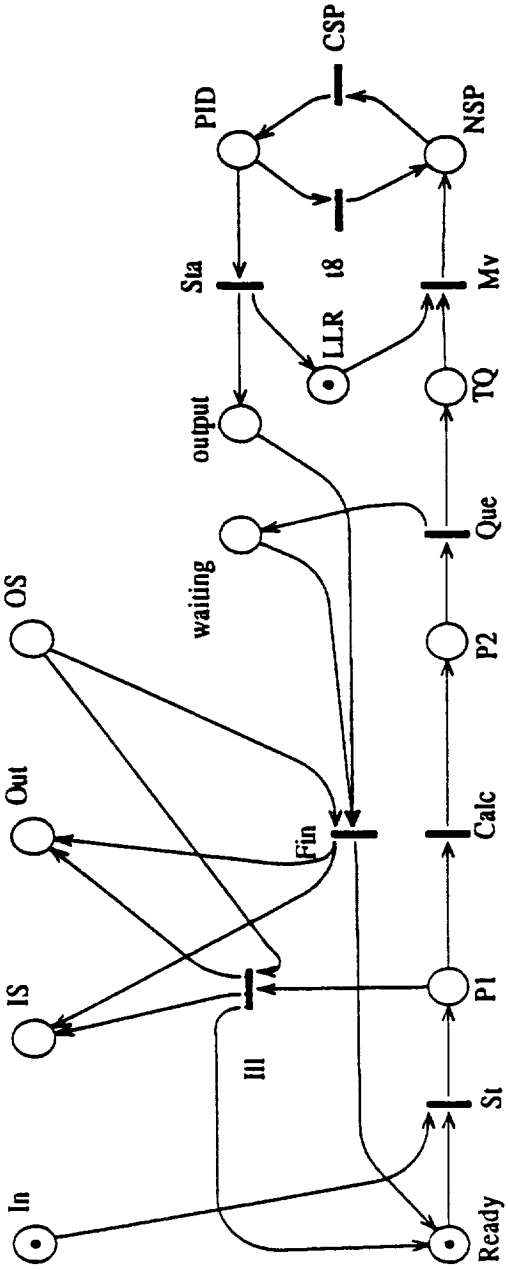

Figure 4.5: The Arm Petri Net Transducer

Goto: Transition for the goto(X, Y, Z) command.

Ill: Illegal location for the arm to go to.

Calc: Calculates the transform for the Cartesian command.

Fin: The Arm has either reached the desired position or an error state.

Que: The motion command is about to be put into the motion Queue.

Mv: Start moving the arm.

CSP: Calculate the next Set Point.

t8: Motion Not yet Completed.

Sta: Fired if in error state, or if the motion has been completed.

Fin: Finished. Returns either "OK" or "Error".

Places:

In: Input Place.

Out: Output Place.

IS, OS: The input and output semaphores.

Ready: Motion system ready for the next command.

PID: This represents the PID loop.

Waiting: The user program is waiting until the lower levels of Kali output either the desired position, or an error.

Output: The Lower levels of Kali have either reached an error, or some position.

LLR: A lower Level Ready, A token here indicated that the Queue is ready to begin the next job in the motion queue.

P1: Test for legality of path.

P2: Ready to enqueue an item.

TQ: Task Queued.

NSP: Gets the Next Set Point.

The subtasks to be processed by the motion coordinator are $\Sigma_m = \{Approach, Move\}$. The translation mappings for the motion coordinator can be defined as follows:

The translation mappings $\sigma_m(Que, Approach)$ can be specified as:

```
set_Kali_mode(CARTESIAN);
load_to_queue(desired_position);
```

The translation mappings $\sigma_m(Que, Move)$ can be specified as:

```
set_Kali_mode(JOINT);
load_to_queue(desired_position);
```

The translation mappings $\sigma_m(CSP, Approach)$ and $\sigma_m(CSP, Move)$ can be specified as:

```
calculate_set_point(current_position, desired_position);
```

All other transitions in the motion coordinator are internal.

4.4 Analysis

4.4.1 Structural Analysis

The Dispatcher and all of the Coordinators are live, thus, the PNT underlying the whole coordination structure is live. Likewise, the net is structurally bounded.

4.4.2 Simulation

Simulations are run to test the communication speed, when the message size and distances between the machines are varied. The default setup is:

Dispatcher Sol, a Sun 4/260, and the system file server, and network host.

Motion Mars, a Sun 3/260.

Vision Venus, a Sun 4/60 SparcStation.

Gripper Earth a Sun 3/150.

Timing of the speed was done by attaching a routine to every transition in the dispatcher, and recording the amount of time passed since that routine was last called. All of the times which measured the delay between sending a message and receiving one back were recorded.

These tests were run with the initial tape:
CalR, *CalV*, *Move*, *Rel*, *Grip*, *Move*, *Approach*, *Rel*, *Look*, *Find*, *Move*, *Find*, *Move*, *Move*, *Approach*.
This results in 52 transitions being fired, resulting in 26 message transmission delays.

4.4.2.1 Message Size Variation

This section tested the the variation in speed when the message size was increased. Under normal operations, sending a token involves the transmission of 8 bytes, and a tape section is 12 to 40 bytes. In this test 50 and 500 bytes were appended to each socket message to see if this deteriorated the speed of the message transmission.

The results of this variation can be seen in figure 4.6. As can be seen adding 50 bytes didn't affect the performance, while adding 500 bytes doubled the expected delay time.

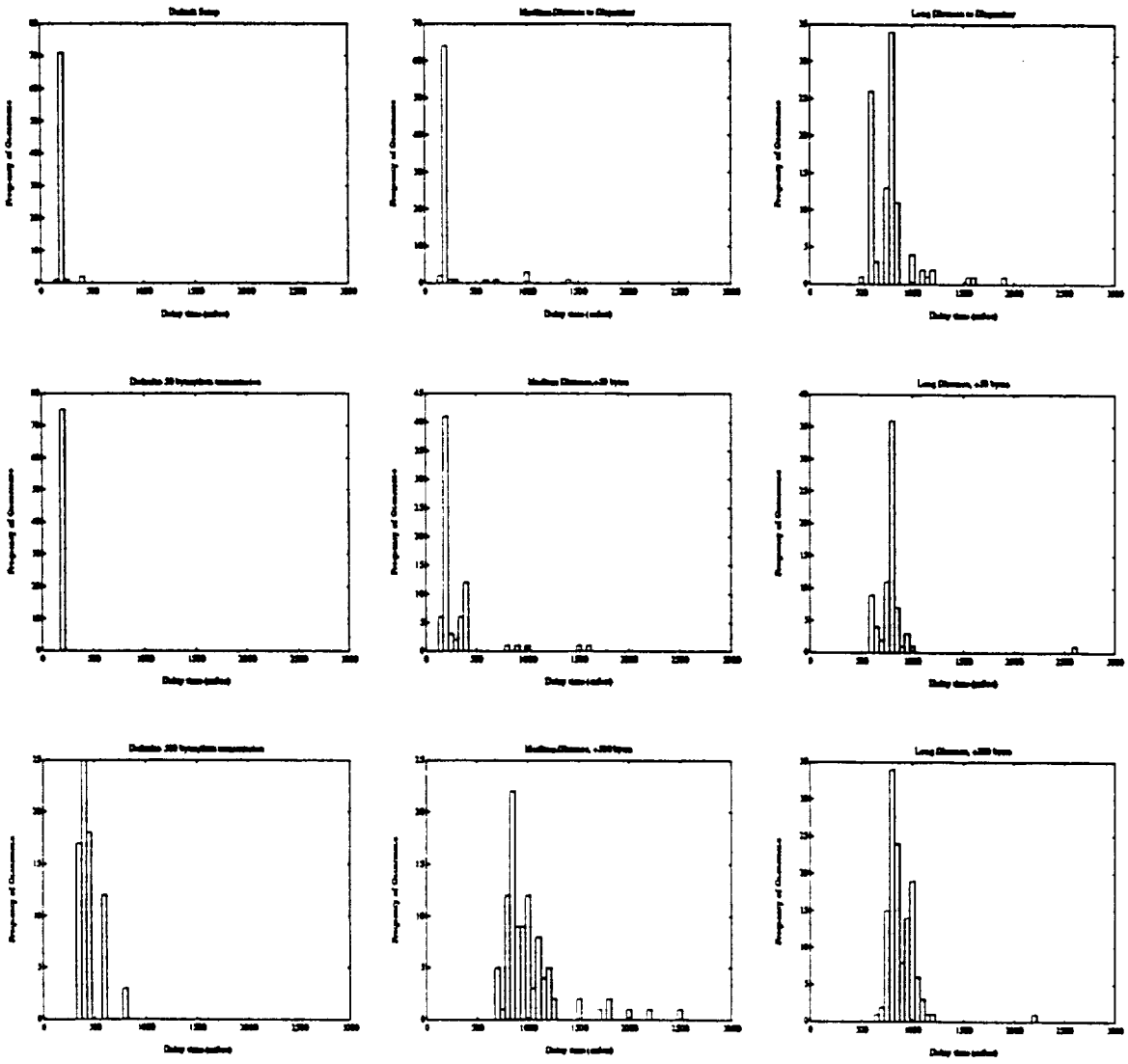

Figure 4.6: Variations in Transmission time with Distance and Data Size

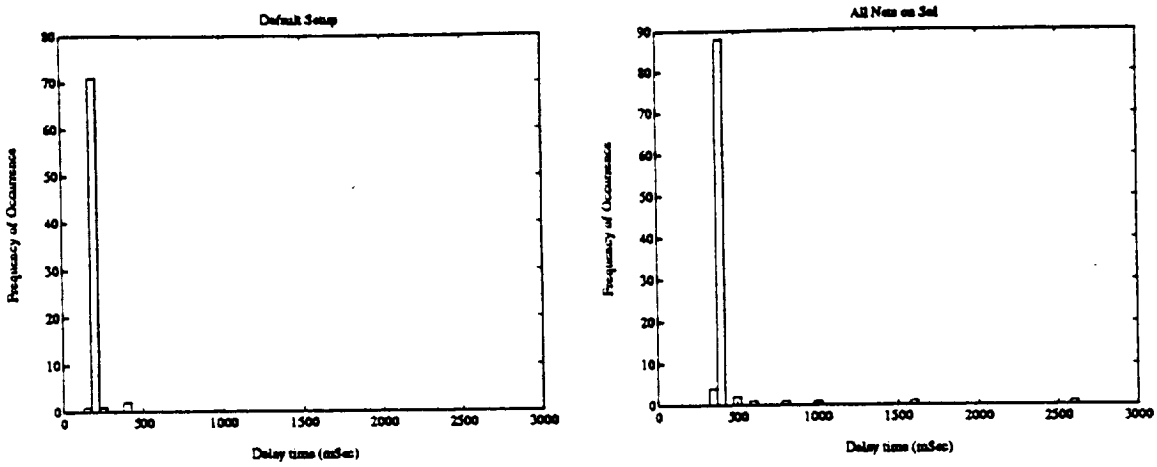

Figure 4.7: Delay increases when sockets are used to communicate on a single machine.

The message sizes were also varied with the computer configuration variation as described in the next section.

4.4.2.2 Computer Configuration Variation

This section tested for variations in speed due to variations in the distances between the computers which were running the nets , and variations in how the computers were connected.

All nets local In this test all of the nets were running on Sol, communicating, as normal, via sockets. The results of this can be seen in figure 4.7.

The explanation for the counter-intuitive increase in delay time can be seen when one realizes that a computer only checks sockets every $200mS$, and since a computer will be synchronized with itself. (and is unlikely to be synchronized with another computer), a round trip socket communication on one computer takes twice as many clock cycles as communication between different computers.

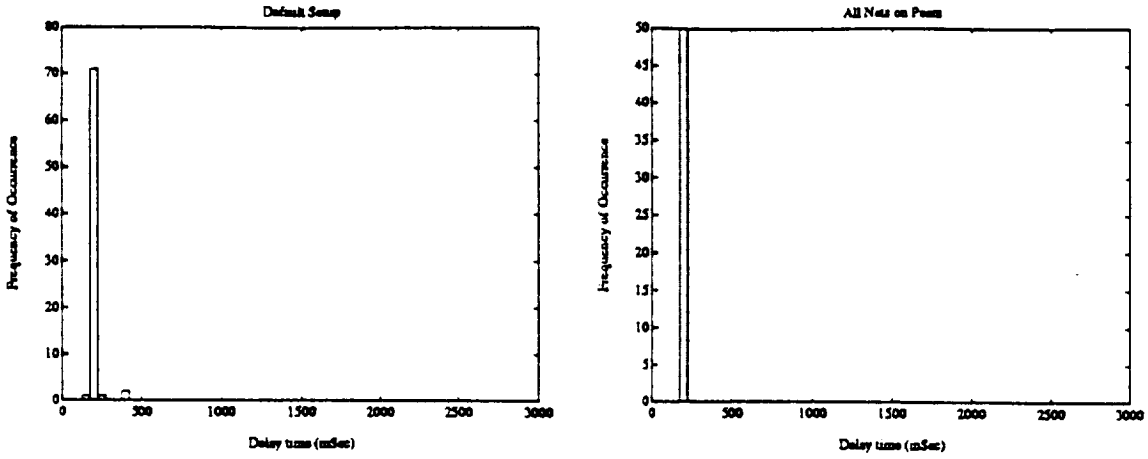

Figure 4.8: There is no particular benefit to placing the Dispatcher on the file server.

On peers In this we moved the dispatcher to Moon (a Sun 4/60GX SparcStation) to see if the dispatcher being the net server improved or worsened the situation.

As can be seen from figure 4.8 there is no significant difference in communication times, thus relieving any necessity of placing the dispatcher on the (possibly overloaded) file server.

Non-local net The dispatcher was placed on pawl3.pawl.rpi.edu, a Sun3/50, located across campus. After testing this, the message size was increased by 50, then 500 bytes.

As can be seen from 4.6 the increased distance produced almost no changes in the normal and +50 byte tests, however when the amount of data transmitted gets larger, the increased distance produces a more pronounced effect. In fact, when the program was run with the display turned on (necessitating increased data transmission to allow the graphics) the program halted with lost data on 3 of the 4 attempts. In further testing it was observed that more displays resulted in less time to failure.

Long distance The dispatcher was located on truebalt.caltech.edu, a Sun4 on based on a Sparc architecture, in Pasadena California. The coordinators were located on their normal computers, resulting in about 3000 miles of Internet communication between the Petri nets.

From figure 4.6 it can be seen that this increased distance resulted in degraded speed even with minimal data. The nets still ran (although more slowly) with 50 extra bytes of data, even when displaying full graphics, however the increase to 500 extra bytes of data per socket transmission resulted in the net failing to complete the task every time (in 6 consecutive trys). The extra load was lowered to 200 bytes which resulted in success in 5 out of the 6 attempts.

CHAPTER 5
CONCLUSIONS AND FUTURE RESEARCH

5.1 Summary and Conclusions

This thesis stated the need for a software tool to assist in the development of distributed hierarchical systems. After a brief overview of Petri nets, available Petri net development tools, and available distributed program design tools it was seen that there were no tools which assisted with the distributed design task which allowed use of all the analysis tooks developed for Petri nets. Petri net transducers were introduced as a model which allowed a full analysis of the nets, and allow control of the transitions fired via a relatively simple mechanism. Because of this they were used as the basis of the TokenPasser program.

With the criteria given, and the additional criteria requiring a friendly output, and the ability to attach routines to each transition (to allow the TokenPasser to develop code which could control more than a mere net), the Token passer program was developed.

The coordination structure for an Intelligent Machine was identified as a project within the domain of the TokenPasser program, and after developing a Petri net model for the Dispatcher and Coordinators of a simple system, TokenPasser was used to develop code to simulate the nets on various machines. The code was then used to test several possible configurations. with respect to how those configurations affected the speed of communication.

This testing allowed several facts to be discovered.

1. Until very large tokens are being passed, the size of the tokens doesn't affect the communication time.

2. The longer the physical distance between the computers the smaller "very

large" is.

3. A computer communicates (via sockets) more slowly with itself than with other computers.

4. Being the file server, and network gateway don't appear to influence communication time.

This thesis also, by developing a PNT for the Coordination Level, and developing code to execute those nets, went one step further in justifying the Coordination theory for the Coordination level presented by Wang [25]. Taken together with the mathematical formulation for the Organization level, and well developed control theory for the Execution level, this further justifies a mathematical theory for Intelligent Machines. Such a mathematical theory will provide a solid foundation for the design, simulation, verification, and implementation of Intelligent Machines.

5.2 Future Research and Modifications

Some useful modifications, and additions to this code are:

- Write program to translate GreatSPN .net files to TokenPasser input files. This will allow easier creation and modification of nets.
- Modify to read .net files, instead of compiling net into code.
- Write communication using Datagram protocol instead of SOCK_STREAM protocol. This should eliminate $200mS$ delay, at a cost of TokenPasser having to do it's own data checking.
- Modify the code to allow arcs which have a weight > 1.
- Give the user a menu of valid commands to append to the tape or insert in the tape at any time, this can be done as on-screen buttons to allow telerobotic operations.

- Add in intermediate level "fast" command which allows a knowledgeable user to watch the progress of the net without the full overhead of a graphical display. (eg. the -semifast option results in no display except for printing out the number of each transition as it fires).

- Rewrite the main routine to allow the user to define the communication among the nets in a more complex manner. Specifically, the current program only allows a two layers of communicating nets, (either a net connects to a pre-existing socket, or it opens a new socket, it can't do both). If the main routine is rewritten to allow both opening and connecting to pre-existing sockets, this will allow the user to define a much more complicated net structure.

- Rewrite to allow for partial display of graphics. Currently a fairly simple net fills the whole screen, it should be possible to mark certain transitions as "uninteresting", allowing the display of a more interesting net by ignoring the trivial places or transitions.

LITERATURE CITED

[1] Al-Jaar, R.Y. and Derochers, A.A. (1987), Petri Nets in Automation and Manufacturing, *Robotics and Automation Laboratory Report*, No. 99, Rensselaer Polytechnic Institute, Troy NY.

[2] Azema, P. et al. (1984) Specification and Verification of Distributed Systems using Prolog Interpreted Petri Nets, *Proc. 7th Int. Conf. Software Eng.*, Orlando USA, 1984, pp. 510-518.

[3] Berthelot, G. and Terrat, R. (1982) Petri Net theory for the Correctness of Protocols, *IEEE Trans. Commun.*, Vol. COM-30, No.12, pp. 2497-2505.

[4] Boehm, B.W. (1976) Software Engineering, *IEEE Transactions On Computers*, Vol c-25, No. 12, pp 1226-1241.

[5] Boehm, B.W., and Papaccio, P.N. (1988), *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, pp. 1462-1476.

[6] Bruno, G. and Marchetto, G. (1986) Process-Translatable Petri Nets for the Rapid Prototyping of Process Control Systems, *IEEE Trans. Software Eng.*, Vol. SE-12, No. 2, pp. 346-357.

[7] Chiola, G. (1985) A Software Package for the Analysis of Generalized Stochastic Petri Nets, *Proceedings of the 1985 Workshop on Timed Petri Nets*, Torino, Italy, pp. 136-143.

[8] Courvoisier, M. Vallette, R. Bigou, J.M. and Esteban, P. (1983) A Programmable Logic Controller based on a high level Specification Tool, *Proc. 1983 Conf. Ind. Electron.*, pp. 174-179.

[9] Crockett, D., Desrochers, A., DiCesare, and Ward, T. Implementation of a Petri Net Controller for a Machining Workstation.(1987), *Proc. 1987 IEEE Int. Conf Robotics Automat.*, pp.1861-1867.

[10] Dugan, J.B., Bobbio, A., Ciardo, G., and Trivedi, K. (1985), The Design of a Unified Package for the Solution of Stochastic Petri Net Models, *Proceedings of the 1985 Workshop on Timed Petri Nets*, Torino, Italy, pp. 6-13.

[11] Feldbrugge, F., Petri Net Toolkit Overview 1989, (1989), *Lecture Notes in Computer Science*, pp. 151-178.

[12] Garg, K. An Approach to Performance Specification of Communication Protocols Using Timed Petri Nets, (1985), *IEEE Trans. Software Eng.*, Vol. SE-11, No. 10, pp.1216-1225.

[13] Johnson, A.M., Malek, M. (1988) Survey of Software Tools for Evaluating Reliability, Availability, and Serviceability *ACM Computing Surveys*, Vol. 20, No. 4, pp. 227-269.

[14] Krogh, B.H., Wilson, R., and Pathak, D. (1988), Automated Generation and Evaluation of Control Programs for Discrete Manufacturing Processes, *Proc IEEE International CIM Conference* Troy, NY, USA, pp. 92-99.

[15] Molloy, M. A CAD Tool for Stochastic Petri Nets, (1986), *Proc. 1986 Fall Joint Computer Conf.*, pp. 1082-1091.

[16] Murata, T., Komoda, N., Masumoto, K., and Haruna, K. (1986), A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and its Applications in Factory Automation.. *IEEE Transactions on Industrial Electronics*, Vol IE-33, pp 1-8.

[17] Nutt, G.J. et al. (1989) OLYMPUS: An interactive Simulation System *1989 Winter Simulation Conference Proceedings*, Washington DC, USA. pp. 601-611.

[18] Peterson, J.L. (1980) A Note on Colored Petri Nets, *Information Processing Letters*, Vol 11, No. 1, pp 40-43.

[19] Peterson, J.L, (1981) *Petri Net Theory and the Modeling of Systems*, Prentice-Hall International, Englewood Cliffs, NJ.

[20] Saridis, G.N. and Stephanou, H.E. (1977) A Hierarchical Approach to the Control of A Prosthetic Arm. *IEEE Trans. on Systems, Man, and Cybernetics*. Vol.SMC-7, No. 6, pp. 407-420.

[21] Saridis, G.N., Foundations of Intelligent Controls, *Proc IEEE Workshop on Intelligent Contr.*, pp 23-27, RPI, Troy N.Y.

[22] Topper, A., Caneshmend, L., and Hayward, V. (1988), A Computing architecture for a Multiple Robot Controller for Space Applications - Kali Project. *Fifth CASI Conference on Astronautics*, Ottowa, 1988.

[23] Wang, F.-Y., Kyriakopoulos K.J., Tsolkas A., and Saridis, G.N. (1990) A Petri Net Coordination Model for an Intelligent Mobile Robot. *CIRSSE Report 50*, RPI, Troy, NY.

[24] Wang, F-Y. and Saridis, G.N. (1990) A Coordination Theory for Intelligent Machines. *IFAC Journal Automatica*, Vol.26, No.9.

[25] Wang, F.-Y. (1990) *A Coordination Theory for Intelligent Machines*, Ph.D. Thesis, ECSE Dept, RPI, Troy, NY.

APPENDIX A
THE TokenPasser CODE

A.1 Compiling Instructions

This is the makefile for the simulation described in this thesis.

```
##
##
##
##                    NOTICE OF COPYRIGHT
##           Copyright (C) Rensselaer Polytechnic Institute.
##                    1990 ALL RIGHTS RESERVED.
##
##
## Permission to use, distribute, and copy is granted ONLY
## for research purposes, provided that this notice is
## displayed and the author is acknowledged.
##
## This software is provided in the hope that it will be
## useful. BUT, in no event will the authors or Rensselaer
## be liable for any damages whatsoever, including any lost
## profits, lost monies, business interruption, or other
## special, incidental or consequential damages arising out
## of the use or inability to use (including but not
## limited to loss of data or data being rendered
## inaccurate or losses sustained by third parties or a
## failure of this software to operate) even if the user
## has been advised of the possibility of such damages, or
## for any claim by any other party.
##
## This software was developed at the facilities of the
## Center for Intelligent Robotic Systems for Space
## Exploration, Troy, New York, thanks to generous project
## funding by NASA.
##
## Package: TokenPasser
##
## File: Makefile
```

```
##
## Written By: Michael Mittmann
##
## Date: 1/30/91
##
## Purpose: The purpose of the package can be found in the file
##    main.c.
##    This file contains the instructions to make 4 routines,
##    which model (by using a PNT) a coordinator controlling
##    an arm, vision system, and gripper.
##
## Modification History:
##
##
disp: draw.o main.o makePet.o petLib.o petri.o postn.o
setup_disp.o \
    transform.o sock_open.o menu.o window_manager.o read_socket.o
    cc -o disp draw.o main.o makePet.o petLib.o petri.o postn.o \
s tup_disp.o transform.o menu.o window_manager.o \
sock_open.o  read_socket.o -lm -lX11

arm : draw.o main.o makePet.o petLib.o petri.o postn.o setup_arm.o
\
    transform.o sock_connect.o menu.o window_manager.o
    read_socket.o
    cc -o arm draw.o main.o makePet.o petLib.o petri.o postn.o \
setup_arm.o transform.o menu.o window_manager.o\
sock_connect.o  read_socket.o -lm -lX11

vision: draw.o main.o makePet.o petLib.o petri.o postn.o
setup_vision.o \
    transform.o sock_connect.o menu.o window_manager.o
    read_socket.o
    cc -o vision draw.o main.o makePet.o petLib.o petri.o postn.o
    \
setup_vision.o transform.o menu.o window_manager.o\
sock_connect.o  read_socket.o -lm -lX11

grip: draw.o main.o makePet.o petLib.o petri.o postn.o
setup_grip.o \
    transform.o sock_connect.o menu.o window_manager.o
    read_socket.o
    cc -o grip draw.o main.o makePet.o petLib.o petri.o postn.o \
```

```
setup_grip.o transform.o menu.o window_manager.o\
sock_connect.o read_socket.o -lm -lX11

window_manager.o: window_manager.c dumb_dec.h
menu.o: menu.c pet.h
draw.o: draw.c xhead.h pltr.h xdraw.h draw.h pet.h dumb_dec.h
main.o: main.c pet.h dumb_dec.h xdraw.h
makePet.o: makePet.c pet.h dumb_dec.h
petLib.o: petLib.c pet.h dumb_dec.h
petri.o: petri.c pltr.h draw.h
postn.o: postn.c pltr.h
setup_disp.o: setup_disp.c pet.h pet2.h dumb_dec.h
setup_grip.o: setup_grip.c pet.h pet2.h dumb_dec.h
setup_arm.o: setup_arm.c pet.h pet2.h dumb_dec.h
setup_vision.o: setup_vision.c pet.h pet2.h dumb_dec.h
transform.o: transform.c pet.h pltr.h dumb_dec.h
sock_open.o: sock_open.c pet.h dumb_dec.h
sock_connect.o: sock_connect.c pet.h dumb_dec.h
read_socket.o: read_socket.c pet.h dumb_dec.h
```

A.2 main.c

This is the main routine.

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
```

```
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: main.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of this package is to provide a facility
**    for translating petri net transducers into runnable code.
**    When compiled with suitable net definition files
**    (see setup_*.c) this package produces code that runs
**    and displays a set of petri nets which communicate with
**    each other via internet sockets.
**    Further instructions, and examples can be found in the
**    directory TokenPasser.doc.
**
**    This file contains the main routine, which:
**        1) determines if windows are to be displayed.
**        2) Sets up the windows.
**        3) forks off a parent to moniter the windows
**        3) calls the socket initilization routines
**        4) calls the Petri net initilization routines
**        5) goes into an infinite loop trying to fire
**           transitions.
**
** Modification History:
**
*/

#include <signal.h>
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "pet.h"
```

```
#include "dumb_dec.h"
#include "xdraw.h"

#define NullArc     (-1)

/*
 * These externs should all be declares in the setup_*.c files
 */
extern int num_transitions,number_of_sockets;
extern struct transition world[];
extern char title[];

/*
 * This extern is declared in read_socket.c
 */
extern int io_came;

int graphics = TRUE;            /* are we doing graphics? */
int should_pause = FALSE;      /* are no transitions enabeled?*/
int redraw_screen = FALSE;     /* User requested redraw? */



/*
 * Data structures needed for X-windows.
 */
Display *display;
Window net_window;
GC gc;
unsigned long foreground, background;

typedef struct {
int    place;      /* True if Place, False if Transition      */
int    used;       /* True if this structure is valid         */
int    type;       /* Indicates if State or Interaction Point */
int    tok;        /* Number of tokens                        */
int    in, out;    /* Number of input and output arcs         */
int    x, y;       /* Grid Location in the Petri-Net window   */
int    party;      /* Module number                           */
char    *name;     /* Name of Place                           */
char    *cond;     /* Condition for Transition                */
char    *code;     /* Action code for the transitions         */
int    priority;   /* Priority of Transition                  */
```

```
} pltrtype;

typedef struct {
int    used;        /* True if this structure is used         */
int    src;         /* Source pltr id for this arc            */
int    dest;        /* Destination pltr id for this arc       */
int    srcnext;     /* Arc id for the succeeding arc from src */
int    destnext;    /* Arc id for the succeeding arc to dest  */
int    y0, x0, y1, x2, y3, x4, y5;
/* Effective Grid coordinates for the arc     */
/* Arc goes through  (x0,y0), (x0, y1), (x2,y1),
 (x2, y3), (x4, y3), (x4, y5)         */
int    off0, off1, off2, off3, off4;
    /* Indicates the offset for each segment of the
       arc on the corresponding grid line         */
} arctype;

int    npltr =140 , narc =140;
pltrtype    pltr[140];
arctype        arc[140];

/* ------------------------------------------------------------
 * redraw_signal()
 * called when a SIGUSR1 signal is sent (which happens when
 * the user requests a redraw.
 *
 * Arguments: None.
 * Returns: Nothing.
 * Requirements: The global variable redraw_screen.
 * ------------------------------------------------------------*/
void redraw_signal(){
redraw_screen = TRUE;
}

/* ------------------------------------------------------------
 * main(argv, argc)
 *
 * The main routine.  It recieves as arguments acommand line
 * variables.  The syntax is currently set up so that if
 * this is the main routine it has a usage of:
 * disp [-fast]
 * and if it is another routine it has a usage of:
 * <cmnd> <hostname> <socketnumber> [-fast]
```

```
*
* This routine follows roughly this pseudocode:
* if(fast option is given) Graphics = false;
* if (Graphics) set up window;
* if (Graphics)
*    {
*     fork();
*     if(process == parent) run window monitering loop;
*     else
*        {
*        initialize sockets;
*        initialize petri_nets;
*        create second type of PN data structure
*        figure out where the arcs should be placed;
*        sit in infinite loop trying to fire transitions;
*        }
*    }
*
* Arguments: argv, argc.  (usage descibed above)
* Returns: Nothing
* Requirements: X11 library, System V enviroment, PF_INET
*    SOCK_STREAM sockets....
* ------------------------------------------------------*/
main(argc, argv)
int argc;
char **argv;
{
    int i,j,good_token,pid,tran_count = 0;

if (strcmp(title, "Dispatcher") == 0){
    if ((argc < 1)||(argc > 2)){
fprintf(stderr, "usage: %s [-fast] \n",argv[0]);
exit(1);
}
    }
else if(argc < 3){
    fprintf(stderr, "usage: %s <hostname><socketnumber>
 [<hostname><socketnumber>...] [-fast] \n",argv[0]);
    exit(1);
    }
if (strcmp("-fast",argv[argc-1]) == 0) graphics = FALSE;
```

```
if (graphics) init_window();

if(graphics && (pid=fork()) != 0){
    XMapRaised (display, net_window);
    event_reading_loop(pid);    /* note that the program won't
    return
     * from this until the user hits q
     * in the window.
     */

    kill(pid,SIGKILL);     /*kill child (laughing like a maniac)*/
    XFreeGC(display, gc);
    XDestroyWindow(display, net_window);
    XCloseDisplay(display);
    exit(0);
    }

else if (graphics){
/* signal to redraw screen */
    signal (SIGUSR1, redraw_signal);

/* reset connection to server for display */
    close(ConnectionNumber(display));
if((display = XOpenDisplay(""))== NULL)
perror("failed XOpenDisplay in child:");

/* open or connect to the sockets  (opening them doesn't use
 * argv or argc                                    */
    sock_start(argc, argv);

/* define the net    */
    make_net();

/* transform to seecondary data structure    */
    fill_pltr_arc_map(world);

/* mark initial conditions */
    (void)initialize_marking();

/* position the arcs */
    for (i=0;i<narc;i++)arc_postn(i);

/* dump on the screen */
```

```
    draw();

/* the infinite loop */
    while(TRUE)
{
should_pause = TRUE;
for (j =0; j< num_transitions;j++)
    {
    if (( good_token = transition_enabled_p(TRAN
    world[j])))
{
fire_transition(TRAN world[j],good_token);
refill_places(FALSE);

/* this line stops the */    if((++tran_count == MAX_NUM_CYCLES)
/* program at some number*/       &&(strcmp(title, "Dispatcher")
== 0))
/* of iterations, delete for*/        exit_program();
/* normal usage. */
should_pause = FALSE;
}
    if (io_came)(void)io_handler(number_of_sockets);
    refill_places(FALSE);
    }
if(redraw_screen) ref_screen();
redraw_screen = FALSE;
if(should_pause) pause();
}
    }
else{
/*
 * We're here iff we're not doing graphics.
 */
/* open or connect to sockets */
    sock_start(argc, argv);

/* make the Petri net */
    make_net();

/* transform to the second data structure */
    fill_pltr_arc_map(world);

/* initialize the net */
```

```
    (void) initialize_marking();

/* place all of the arcs */
    for (i=0;i<narc;i++)arc_postn(i);

/* start infinite loop */
    while(TRUE)
{
should_pause = TRUE;
for (j =0; j< num_transitions;j++)
    {
    if (( good_token = transition_enabled_p(TRAN
    world[j])))
{
fire_transition(TRAN world[j],good_token);

/*This line stops the program*/    if((++tran_count ==
MAX_NUM_CYCLES)
/*when 52 transitions have */       &&(strcmp(title, "Dispatcher")
== 0))
/* fired, remove for normal use */    exit_program();

should_pause = FALSE;
}
    if(io_came)(void)io_handler(number_of_sockets);
    }
if(should_pause) pause();
}
    }
}


/*-------------------------------------------------------
 * Author's note:
 *
 * running different sets of code depending on if graphics
 * is true or false is very ugly.  This allows the
 * possibility of the net running differently if a
 * modification is made to one loop but not to the other.
 *
 * The only way I could see of fixing this was to put several
 * extra if statements into the loop.  I didn't like this
 * solution, but given the fact that most of the delay comes
```

```
 * from the socket communication, the effects of this
 * modification may be negligable.
 * ------------------------------------------------------*/
```

A.3 draw.c

This file contains the drawing routines.

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: draw.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
```

```
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains the routines associated with the
**    drawing and redrawing of the Petri net on the
**    Xwindow.
**
** Modification History:
**
*/
/*-------------------------------------------------------------
 * draw.c:
 *    Handles all the drawing of Places/Transitions
 *    /Arcs/Hilighting in the Petri-Net window.
 *-------------------------------------------------------------*/
#include <stdio.h>
#include <math.h>
#include "xhead.h"
#include "pltr.h"
#include "xdraw.h"
#include "draw.h"
#include "pet.h"
#include "dumb_dec.h"


#define    MAX_PLACES    70    /* PUT IN INCLUDE FILE */
#define FLASHTIME    150000

extern int num_transitions;

#define dMark    8
#define dArcAng    20
#define SIZE_RECT_X ((int)(GridSize * 1.6))
#define SIZE_RECT_Y GridSize
#define RECT_X 0
#define RECT_Y    0
#define    Dia    GridSize
/*-------------------------------------------------------------
 * mag(x, off) Converts the Grid Coordinates to the real
 * coordinates on the screen. x is the grid number, off is
 * the offset on that grid.
 *-------------------------------------------------------------*/
#define mag(x, off)    ((x * GridSize + off * OffSize + GridOff))
```

```
/*-------------------------------------------------------------
 * draw()
 *    Draws all the Places, Transitions and Arcs.
 *    Also fills the places, and draws the "tape".
 * Arguments: None
 * Returns: Nothing
 * Requirements: This requires an opened X-window, and filled
 *    pltr, and arc data structures.
 *-------------------------------------------------------------*/
void draw()
{
    int    i;

    for(i=0; i<npltr; i++)
if (pltr[i].used) {
    if (pltr[i].place)
dr_place(i);
    else
dr_trans(i);
}
    for(i=0; i<narc; i++)
if (arc[i].used)
    dr_arc(i);
    draw_tape();
    refill_places(TRUE);
}


/*-------------------------------------------------------------
 * draw_tape()
 * Draws the menu tape..
 * Note that it clears the area first, so that the words
 * aren't overwriting anything else and are legible.
 * Arguments: None.
 * Returns: Nothing.
 * Requirements: An open X-window, and the routines in menu.c
 *-------------------------------------------------------------*/
void draw_tape()
{
int i;
```

```
    for(i=0;i<SIZE_MENU;i++)
{
draw_tape_box(i);
}

/* erase all previous arrows... */
    st_erase();
    for(i=0;i<SIZE_MENU;i++)
{
XFillArc(display, net_window, gc, SIZE_RECT_X/2 +
i*SIZE_RECT_X,
    SIZE_RECT_Y, Dia/2, Dia/2, 64*60, 64*240);
}
/* and draw an arrow pointing to the current menu selection */
    st_normal();
    XFillArc(display, net_window, gc, SIZE_RECT_X/2 +
    current_head_of_tape()
* SIZE_RECT_X, SIZE_RECT_Y, Dia/2, Dia/2, 64*240, 64*60);
}

/*-----------------------------------------------------------------
 * draw_tape_box()
 * This draws an individual tape box.
 * Arguments: i, an integer specifying which box is to be drawn
 * Returns: Nothing.
 * Requirements: An open X window, and the routines in menu.c
 *-----------------------------------------------------------------*/
void draw_tape_box(i)
int i;
    {
    char *string;

    st_erase();
    XFillRectangle(display, net_window, gc, RECT_X +
i*SIZE_RECT_X, RECT_Y, SIZE_RECT_X, SIZE_RECT_Y);
    st_normal();
    XDrawRectangle(display, net_window, gc, RECT_X +
i*SIZE_RECT_X, RECT_Y, SIZE_RECT_X, SIZE_RECT_Y);
    string= get_menu_string(i);
    XDrawString(display, net_window, gc, RECT_X+i*SIZE_RECT_X,
    SIZE_RECT_Y /2, string, strlen(string));
    }
```

```
/*-------------------------------------------------------------
 * dr_place(i)
 *    Draws the place, stored in pltr[i].
 *    A Circle is drawn, with tokens inside and its name,
 *    truncated to 5 chars is drawn next to the circle, along
 *    with its id number (to be used in later verification.)
 * Arguments: i, an integer specifying which place to draw.
 * Returns: Nothing.
 * Requirements: An open X-window, and a filled pltr data
 *        structure.
 *-------------------------------------------------------------*/
void dr_place(i)
int    i;
{
    int    x, y;
    char    name[10];

    x = mag(pltr[i].x, 0);
    y = mag(pltr[i].y, 0);

    XDrawArc (display, net_window, gc,
x - Dia/2 , y - Dia/2,
Dia, Dia,
0, 360 *64);

    sprintf(name, "%2d:", i);
    if (pltr[i].name != NULL)
strncat(name, pltr[i].name, 5);

    XDrawString(display, net_window, gc,
x + 2*Dia/4, y,
name, strlen(name));

}

/*-------------------------------------------------------------
 * empty_place(i)
 * This routine draws a white filled circle in a place,
 * effectively erasing all of the tokens that were drawn there.
 * Arguments: i, an integer specifying which place to empty.
 * Returns: Nothing
 * Requirements: An open X-window, and a full pltr data structure
 *-------------------------------------------------------------*/
```

```
empty_place(i)
int    i;
{
    int    x, y;

    x = mag(pltr[i].x, 0);
    y = mag(pltr[i].y, 0);

    st_erase();
    XFillArc (display, net_window, gc,
x-(Dia-2)/2 , y - (Dia -2)/2,
Dia-2, Dia-2,
0, 360 *64);
    st_normal();
}



/* ------------------------------------------------------------
 * refill_places(new)
 * This routine draws tokens in the places.
 *
 * If new== TRUE then it redraws all of the tokens, otherwise
 * it only redraws the tokens in places where the number of
 * tokens has changed.
 *
 * This routine also takes care of redrawing the "tape"
 *
 * Arguments: A boolean specifying if all places which have
 *    tokens are to be redrawn, or only the ones which
 *    changed their number of tokens.
 * Returns: Nothing.
 * Requirements: An open X-window, a full pltr data structure,
 *    and the routines in menu.c
 * ------------------------------------------------------------*/

void refill_places(new)
int new;
{
    static int has_tokens[MAX_PLACES] = {0};  /* record of # of
    tokens
       * in each place the
       last
```

```
       * time this routine
       was
       *
       called.
       */
    static int last_head_position = 0;
    int i,j;
    char *string;

/* checking each place, redraw the tokens if needed */
    for (i = 0; i < MAX_PLACES;i++)
{
if (pltr[i+num_transitions].used)
    {
    if ((has_tokens[i] < pltr[i+num_transitions].tok)||
    new)
{
empty_place(i+num_transitions);
for (j = pltr[i+num_transitions].tok; j>0;j--)
    drawmark(pltr[i+num_transitions].x,
pltr[i+num_transitions].y,j);
has_tokens[i] = pltr[i+num_transitions].tok;
}
    else if((has_tokens[i] >pltr[i+num_transitions].tok)||
    new)
{
empty_place(i+num_transitions);
if ((has_tokens[i] = pltr[i+num_transitions].tok
    )!=0)
for (j = pltr[i+num_transitions].tok; j>0;j--)
    drawmark(pltr[i+num_transitions].x,
pltr[i+num_transitions].y,j);
}
    }
}
/* if the pointer for the head of the tape has been incremented,
redraw
 * the tape
 */
    if((i= current_head_of_tape()) != last_head_position)
{
st_erase();
XFillArc(display, net_window, gc, SIZE_RECT_X/2 +
```

```
    last_head_position* SIZE_RECT_X,
     SIZE_RECT_Y, Dia/2, Dia/2, 64*240, 64*60);
XFillRectangle(display, net_window, gc, RECT_X +
    last_head_position*SIZE_RECT_X +1 , RECT_Y +1 ,
    SIZE_RECT_X -2 , SIZE_RECT_Y -2 );

st_normal();
    string= get_menu_string(last_head_position);
    XDrawString(display, net_window, gc, RECT_X+
last_head_position*SIZE_RECT_X,SIZE_RECT_Y /2,
 string, strlen(string));
XFillArc(display, net_window, gc, SIZE_RECT_X/2 + i
    * SIZE_RECT_X, SIZE_RECT_Y, Dia/2, Dia/2,
    64*240, 64*60);
last_head_position = i;
}
XFlush(display);
}



/*-------------------------------------------------------------
 * angoff(x, ang)
 *    (angle offset) is used to calculate to place the
 *     tokens inside a place.
 *
 * Arguments: x, an integer specifying which token this is,
 *    ang, the desired offset/token (in degrees)
 * Returns: A double spcifying how many radians off of the base
 *    direction the token should be placed.
 * Requirements: M_PI is in math.h
 *-------------------------------------------------------------*/
double
angoff(x, ang)
int    x, ang;
{
    double res;

    res = x * ang * M_PI / 180.0;
    return res;
}
```

```
/*-------------------------------------------------------------
 * drawmark(x, y, n)
 *    draws Nth token at (x,y).  Tokens are drawn in a
 *    circular fashion inside the Place. The first token
 *    is placed in the center, and the rest are drawn in a
 *    counterclockwise fashion, only one round. If too many
 *    tokens are there, they may overlap.
 *
 * Arguments: x,y integers, specifying the location of the
 *    place in which the token is to be drawn.  n, an
 *    integer specifying which token it is.
 * Returns: Nothing.
 * Requirements: An open X window.
 *-------------------------------------------------------------*/
void drawmark(x, y, n)
int x, y, n;
{
    int    xx, yy;

    xx = mag(x, 0);
    yy = mag(y, 0);
    n--;
    if (n != 0) {
xx += dMark * cos(angoff(n, 3*dArcAng));
yy -= dMark * sin(angoff(n, 3*dArcAng));
    }
    if(n>=0)
{
XFillArc(display, net_window, gc,
    xx - dMark/2, yy - dMark/2,
    dMark, dMark,
    0, 360*64);
}
}


/*-------------------------------------------------------------
 * dr_trans(i)
 *    Draws the transition (a horizontal line, followed
 *    by its id) stored in "pltr[i]";
 *
 * Arguments: i, an integer spcifying which transition.
 * Returns: Nothing.
```

```
 * Requirements: An open X-window, and the pltr data structure.
 *-------------------------------------------------------------*/
void dr_trans(i)
int    i;
{
    int    x, y;
    char    name[3];

    x = mag(pltr[i].x, 0);
    y = mag(pltr[i].y, 0);

    XDrawLine(display, net_window, gc,
x - Dia/2, y,
x + Dia/2, y);
    sprintf(name, "%2d:", i);

    XDrawString(display, net_window, gc,
x + 2*Dia/4, y,
name, strlen(name));
}


/*-------------------------------------------------------------
 * dr_arc(i)
 *    draws an arc, stored in arc[i]. The five segments
 *    of the arc are drawn as straight lins. The Arrow mark
 *    at the end of the arc is created as a small filled arc.
 *
 * Arguments: i, the number of the arc.
 * Returns: Nothing
 * Requirements: An open X-window, and the arc data structure.
 *-------------------------------------------------------------*/

void dr_arc(i)
int    i;
{
    XPoint    l[6];

    l[0].x = mag( arc[i].x0, arc[i].off0 );
    l[0].y = mag( arc[i].y0, 0 );
    if (pltr[arc[i].src].place)
l[0].y += Dia/2;
```

```
    l[1].x = l[0].x;
    l[1].y = mag( arc[i].y1, arc[i].off1 );

    l[2].x = mag( arc[i].x2, arc[i].off2 );
    l[2].y = l[1].y;

    l[3].x = l[2].x;
    l[3].y = mag( arc[i].y3, arc[i].off3 );

    l[4].x = mag( arc[i].x4, arc[i].off4 );
    l[4].y = l[3].y;

    l[5].x = l[4].x;
    l[5].y = mag( arc[i].y5, 0 );
    if (pltr[arc[i].dest].place)
l[5].y -= Dia/2;

    XDrawLines(display, net_window, gc,
l, 6, CoordModeOrigin);

    XFillArc(display, net_window, gc,
l[5].x-Dia/4, l[5].y-Dia/4,
Dia/2, Dia/2,
(45)*64, 90*64);

}

/* ---------------------------------------------------------
 * A routine used to hilight a place or transition for a
 * short time, then unhilight it.  Note, for some reason
 * GXinvert works, while GXxor doesn't.
 *
 * Arguments: i, the number of the transition
 * Returns: Nothing
 * Requirements: The pltr data structure, and an open X-window
 * ---------------------------------------------------------*/
void flash_pltr(i)
int i;
{
    int x,y;

     XSetFunction(display, gc, GXinvert);
    x = mag(pltr[i].x, 0);
```

```
    y = mag(pltr[i].y, 0);

    XFillArc (display, net_window, gc,
x - Dia/2 , y - Dia/2,
Dia, Dia,
0, 360 *64);

    XFlush(display);
    usleep((unsigned)FLASHTIME);

    XFillArc (display, net_window, gc,
x - Dia/2 , y - Dia/2,
Dia, Dia,
0, 360 *64);
    XFlush(display);
    XSetFunction(display, gc, GXcopy);
}


/*-------------------------------------------------------------
 * st_erase()
 *    changes the foreground to white so that anything
 *     can be overwritten.
 *
 * Note that this is being used because
 *    XSetFunction(display, gc, GXClear); didn't seem to work...
 *
 * Arguments: None
 * Returns:Nothing
 * Requirements: The display data structure.
 *-------------------------------------------------------------*/
void st_erase()
{
    XSetForeground(display, gc,
    WhitePixel(display,DefaultScreen(display)));
}

/*-------------------------------------------------------------
 * st_normal()
 *    (set normal mode), undoes the effect of st_erase, if
 *    used earlier.  Using st_normal multiple times should
 *    not cause any harm.
 *
```

```
 * Arguments: None
 * Returns: Nothing.
 * Requirements: The display data structure.
 *-------------------------------------------------------------*/
void st_normal()
{
    XSetForeground(display, gc,
    BlackPixel(display,DefaultScreen(display)));
}

/*  -------------------------------------------------------------
 * void ref_screen()
 * Redraws the whole screen.
 *
 * Arguments: None
 * Returns: Nothing
 * Requirements: The display and net_window data structures.
 * -------------------------------------------------------------*/
void ref_screen()
{
    XClearWindow(display, net_window); /* clears the petri-net
    window */
    draw();
}
```

A.4 makepet.c

This file contains the routines needed to produce the data structures defining net.

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
   displayed and the author is acknowledged.

  This software is provided in the hope that it will be
```

```
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: makePet.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**     main.c.
**     This file contains routines to fill the data structures
**     defining the net.
**
** Modification History:
**
*/

#include <varargs.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include "pet.h"
#include "dumb_dec.h"


extern int sock_send();
extern struct transition world[];
extern struct place_ptr place_ptr_array[];
```

```
extern int menu[];

/* ------------------------------------------------------
 *  place_place()
 * This routine is used to declare the location of a place
 *  on the graphic display...
 *
 * Arguments: place_num, an integer specifying a place.
 *    x_loc, y_loc, the locations on the screen.
 * Returns: Nothing
 * Requirements: None.
 * ------------------------------------------------------ */
void place_place(place_num,x_loc, y_loc)
int x_loc, y_loc,place_num;
{
if ((x_loc % 2 == 1) ||(y_loc % 2 == 1))
    printf("warning the location of place %d is odd
    \n",place_num);
place_ptr_array[place_num].x_loc = x_loc;
place_ptr_array[place_num].y_loc = y_loc;
}


/* ------------------------------------------------------
 * identity (value)
 * this is the default routine used for the coloration
 * and decoloration of tokens.  It doesn't change the
 * token value.
 *
 * Arguments: value: and integer.
 * Returns: value: the same integer.
 * Requirements: reflexive identity property.
 * ------------------------------------------------------*/
int identity(value)
int value;
{
    return(value);
}

/* ------------------------------------------------------
 * initialize_transition(fcn_ptr,init_tran,x_loc, y_loc)
 *
 * This function points both the preplaces and the
```

```
 * post places of a transition to NULL, and points
 * the transition to the function specifies in the call.
 *
 * It should be called with a call like:
 * initialize_transition (function_name, pointer_to_transition);
 *
 * Arguments: fcn_ptr: a pointer to the function which
 *        the transition fires.
 *    init_tran: a pointer to the transition
 *        being initialized.
 *    x_loc, y_loc, the x and y screen locations.
 * Returns: Nothing
 * Requirements: The obvious.
 * ------------------------------------------------------*/
void initialize_transition(fcn_ptr,init_tran,x_loc, y_loc)
int (*fcn_ptr)();
struct transition *init_tran;
{
    init_tran->routine = fcn_ptr;
    init_tran->pre_places = NULL;
    init_tran->post_places = NULL;
    init_tran->enabled_by = NULL;
    init_tran->menu_requirements = NULL;
    init_tran->consumes_menu = TRUE;
    if ((x_loc % 2 == 1) || (y_loc %2 == 1))
printf("warning the location of transition %d is odd\n"
    ,init_tran - &(world[0]));
    init_tran->x_loc = x_loc;
    init_tran->y_loc = y_loc;

}



/* ------------------------------------------------------
 * void  add_pre_list(va_alist)
 *
 * This function should be called with a call like:
 * add_pre_list(&(transition.pre_place), &int1, &int2 ,NULL);
 * to add integers int1 and int2 to the precurser list for
 * transition
 *    (note transition is a data structure of type transition)
 *
```

```
 * this takes a pointer to a transition, and a list of
 * pointers to integers and forms a linked list of pointers
 * to integers starting with the pointer pre_places in the
 * specified transition.
 * For some clarification see varargs(3), which will clarify the
 variable
 * number of arguments...
 *
 * Arguments: a list of pointers as described above.
 * Returns: Nothing
 * Requirements: Link in var_args.h
 * ------------------------------------------------------*/
void  add_pre_list(va_alist)
va_dcl
{
    va_list pointer;
    struct pre_pointer *new_place, **old_place;
    struct place_ptr *place_header;

    va_start(pointer);
    old_place = va_arg(pointer, struct pre_pointer **);
    while ((place_header=va_arg(pointer,struct place_ptr *)) !=
    NULL)
{
new_place = (struct pre_pointer *)
    malloc(sizeof(struct pre_pointer));
if (new_place == NULL)
    {perror ("add_preplace:malloc:");
     exit(-1);
    }
new_place->place = place_header;
new_place->next_place = NULL;
new_place->decoloration_routine = identity;
*old_place = new_place;                /* point the
previous */
old_place = &(new_place->next_place);
/*new_place->next_place*/
}                                      /* at the current
place*/
    va_end(pointer);
}
```

```
/* ------------------------------------------------------
 *  add_post_list(va_alist)
 * This function should be called with a call like:
 * add_post_list(&(transition.post_place),&int1,rv1,
 *        &int2,rv2,&int3,rv3,NULL);
 * to add integers int1 int2 and int3 to the post
 * list for transition
 *    (note transition is a data structure of type
 *     transition, and rv are integers which = 0 if the
 *    data is local, and = the socket number where the
 *    data belongs if the data isn't local.)
 *
 * this takes a pointer to a transition, and a list of
 * pointers to integers and forms a linked list of pointers
 * to integers starting with the pointer post_places in the
 * specified transition.
 * For some clarification see varargs(3), which will clarify the
 variable
 * number of arguments...
 *
 * Arguments: a list of pointers as described above.
 * Returns: nothing
 * Requirements: Include var_args.h
 * ------------------------------------------------------ */

void add_post_list(va_alist)
va_dcl
{
    va_list pointer;
    struct post_pointer *new_place, **old_place;
    struct place_ptr *place_header;

    va_start(pointer);
    old_place = va_arg(pointer, struct post_pointer **);
    while ((place_header= va_arg(pointer, struct place_ptr *)) !=
    NULL)
{
new_place = (struct post_pointer *)
    malloc(sizeof(struct post_pointer));
if (new_place == NULL)
    {perror ("add_postplace:malloc:");
     exit(-1);
    }
```

```
new_place->remote = va_arg(pointer, int);
new_place->place = place_header;
new_place->coloration_routine = identity;
new_place->data_trans_routine = sock_send;
new_place->next_place = NULL;
*old_place = new_place;                /* point previous to
*/
old_place = &(new_place->next_place);        /*this one
*/
}
    va_end(pointer);
}


/* ------------------------------------------------
 *  instruct_in_menu_p(trans)
 *
 * this routine checks to see if there is a list of
 * "menu" commands which are required for the
 * transitions to be fired.  If the transition requires
 * a menu command, this routine goes through the list
 * of possible enabling commands, and checks to see if
 * any of them match the current command.
 *
 * Arguments: a pointer to the transition to be checked.
 * Returns: True/False
 * Requirements: the routines in menu.c
 * ------------------------------------------------------*/
int instruct_in_menu_p(trans)
struct transition *trans;
    {
    int men_val;
    struct enabelors *cur_menu_allowed;

    if (trans->menu_requirements == NULL)
return (TRUE);
    if ((men_val = get_menu_value()) == NO_MENU_CMND)
 return(FALSE);
    for (cur_menu_allowed= trans->menu_requirements;
cur_menu_allowed!=NULL;
cur_menu_allowed = cur_menu_allowed->next_enabelor)
{
if (cur_menu_allowed->enabelor== men_val)
```

```
    return(TRUE);
}
    return (FALSE);
    }


/* ------------------------------------------------
 * add_token(place_num, data)
 *
 * This routine adds a token to the list of tokens attached
 * to a place, and calls a routine that adds one to the
 * number of tokens in the pltr represnetation.
 *
 * Arguments: place_num: integer declareig whaigh place
 *    gets the token.
 *        data: integer = the value of the token.
 * Returns: Nothing.
 * Requirements: the pltr and normal data structures.
 * ------------------------------------------------*/
void add_token(place_num, data)
int data, place_num;
    {
    struct place_contains *newplace, *temp;

    if ((newplace = (struct place_contains *)
malloc(sizeof (struct place_contains))) == NULL)
{
perror("add_token:malloc:");
exit(0);
}
    temp = place_ptr_array[place_num].place;
    place_ptr_array[place_num].place =newplace;
    newplace->object = data;
    newplace->next_contents = temp;

 /* putting tokens in the drawing representation.... */
    (void) add_tok_in_pltr_rep(place_num);
    }


 /* ------------------------------------------------
  * remove_token (place_num, data,trans,decoloration_routine)
  *
```

```
* This routine searches through the data in place place_num,
* and if either:
*     it can find a token such that:
*     decoloration_routine(token) == data
* or:
*     the transition is enabeled by any token, and it
*     finds a token.
* It removes that token, and returns OK.
* If this somehow fails, the routine returns an error.
*
* Arguments: place_num: the place which has the token.
*        data: the value of the token to be removed.
*        trans: the transition which may have an
*    enabeling requiremment.
*        decoloration_routine: a pointer.
* Returns: OK/ERROR
* Requirements: Just all of the data structures.
* ------------------------------------------------------*/
remove_token (place_num, data,trans,decoloration_routine)
int data, place_num;
struct transition *trans;
int (*decoloration_routine)();
    {
    struct place_contains *current, **previous;
    int any_data;

    any_data = (trans->enabled_by == NULL);
    previous = (struct place_contains **)
    &(place_ptr_array[place_num]);
    for (current = place_ptr_array[place_num].place; current !=
    NULL;
current = current->next_contents)
{
if (decoloration_routine != NULL)
    {
    if ((data ==
    (*decoloration_routine)(current->object))||
any_data)
{
*previous = current->next_contents;
free ((char *) current);
(void) remove_tok_in_pltr_rep(place_num);
return(OK);
```

```
}
    else previous = &(current->next_contents);
    }
else
    {
    if ((data == current->object) || any_data)
{
*previous = current->next_contents;
free ((char *) current);
remove_tok_in_pltr_rep(place_num);
return(OK);
}
    else previous = &(current->next_contents);
    }
}
    return (ERROR);
    }

/* ---------------------------------------------------------
 * declare_enab_tokens(va_alist)
 *
 * This is a function which declares either the menu value
 * required to fire a transition, or the token values
 * required to fire a (colored) transition.
 *
 * This function should be called with a call like:
 * declare_enab_tokens(transition->(whichever), int1,
 *      int2 ,int3, NULL);
 * to add integers int1 int2 and int3 to the enabelor list
 * for transition
 *    (note transition is a data structure of type transition)
 *     ((whichever) is either enabeled_by or menu_requirements)
 *
 * this takes a pointer to a enabelor and a list of pointers
 *  to integers and forms a linked list of pointers to integers
 * starting with the pointer enabelor in the specified
transition.
 * For some clarification see varargs(3), which will
 * clarify the variable number of arguments...
 *
 * Arguments: a list of pointers as described above.
 * Returns: nothing
 * Requirements: Include var_args.h
```

```
 * ------------------------------------------------------*/
void declare_enab_tokens(va_alist)
va_dcl
{
    va_list pointer;
    struct enabelors *new_one, **old_one;
    int enab_tokens;

    va_start(pointer);
    old_one= va_arg(pointer, struct enabelors **);
    while ((enab_tokens=va_arg(pointer,int )) != NULL)
{
new_one= (struct enabelors *)
    malloc(sizeof (struct enabelors));
if (new_one == NULL)
    {perror ("declare_enabelor_tokens:malloc:");
     exit(-1);
    }
new_one->enabelor= enab_tokens;
new_one->next_enabelor= NULL;
*old_one= new_one;                    /* point the previous
*/
old_one= &(new_one->next_enabelor);
/*new_place->next_place*/
}                                    /* at the current
place*/
    va_end(pointer);
}

/* ------------------------------------------------------
 * fill_post_pntr(place, tran, data_trans, coloration)
 *
 * This routine puts some addional information into the
 * the post_pointer between the given place and transition.
 *
 * This is used for declaring a particular data_transition
 * routine (other than the default (sock_send) or a
 * coloration routine.
 *
 * Arguments: place, tran: integers referring to the place
 *     and transition which need the infiormation attached.
 *        date_trans, coloration, the routines which
 *    will get attached.
```

```
 * Returns: True/False (depending on sucess)
 * Requirements: Just the data structures.
 * ------------------------------------------------------*/
int fill_post_pntr(place, tran, data_trans, coloration)
int place, tran;
int (*data_trans)(), (*coloration)();
    {
    struct post_pointer *temp;
    int sucess;

    sucess = FALSE;
    for (temp = world[tran].post_places; temp != NULL;
temp = temp->next_place)
{
if ((temp->place - &place_ptr_array[0])== place)
    {
    temp->data_trans_routine = data_trans;
    temp->coloration_routine = coloration;
    sucess = TRUE;
    }
}
    return(sucess);
}

/* ------------------------------------------------------
 * fill_pre_pntr(place, tran, coloration)
 *
 * This routine puts some addional information into the
 * the pre_pointer between the given place and transition.
 *
 * This is used for declaring a particular coloration routine.
 *
 * Arguments: place, tran: integers referring to the place
 *     and transition which need the infiormation attached.
 *        coloration, the routines to be attached.
 * Returns: True/False (depending on sucess)
 * Requirements: Just the data structures.
 * ------------------------------------------------------*/
int fill_pre_pntr(place, tran, coloration)
int place, tran;
int  (*coloration)();
    {
    struct pre_pointer *temp;
```

```
    int sucess;

    sucess = FALSE;
    for (temp = world[tran].pre_places; temp != NULL;
temp = temp->next_place)
{
if ((temp->place - &place_ptr_array[0])==place)
    {
    temp->decoloration_routine = coloration;
    sucess = TRUE;
    }
}
    return(sucess);
}
```

A.5 menu.c

This file contains the routines needed to control and manipulate the "tape".

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
```

```
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: menu.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains the routines to manage the "tape"
**    for the PNT
**
** Modification History:
**
*/

#include "pet.h"
extern int menu[];
int head = 0;
int tail = 0;

/* ---------------------------------------------------------
 * get_menu_string(menu_slot_number)
 * This routine returns a string associated with the tape
 * command in tape location menu_slot_number.
 *
 * Arguments: menu_slot number, an integer, specifying
 *    which menu slot.
 * Returns: a character string associated with the command
 *     in that slot.
 * Requirements: none.
 * ---------------------------------------------------------  */
char *get_menu_string(menu_slot_number)
int menu_slot_number;
    {
static char *menu_strings[] ={"empty ",/* Note that this list
should */
      "CalR  ",/* be in the same order
```

```
     as    */
     "Move  ",/* list in pet.h */
     "Approa",
     "CalV  ",
     "Grip  ",
     "Rel   ",
     "MeaFo ",
     "GoFor ",
     "MeaPo ",
     "GoPos ",
     "Cross ",
     "Retur ",
     "Look  ",
     "Find  ",
     "Conti ",
     "Slave "};

   return (menu_strings[menu[menu_slot_number]]);
   }


/* ------------------------------------------------------
 * current_tail_of_tape()
 * This routine esists in case anyone needs to know if
 * new commands have been written to the tail of the tape..
 *
 * Arguments:none
 * Returns:integer location of tail.
 * Requirements: none
 * ------------------------------------------------------*/
int current_tail_of_tape()
    {
    return(tail);
    }

/* ------------------------------------------------------
 * current_head_of_tape
 * This routine is normally used so that outside routines
 * can tell if tape commands have been removed.
 *
 * Arguments:none
 * Returns: integer specifying location of head of tape.
 * Requirements: none
```

```
 * ------------------------------------------------------*/
int current_head_of_tape()
    {
    return(head);
    }

/* ------------------------------------------------------
 * get_menu_value()
 * This returns the value of the command currently at
 * the head of the tape
 *
 * Arguments: none
 * Returns: int. value of command at head of tape, or error.
 * Requirements: none
 * ------------------------------------------------------*/
int get_menu_value()
    {
    if (head != tail)
    return (menu[head]);
    return (NO_MENU_CMND);
    }


/* ------------------------------------------------------
 * insert_menu_command_immediate(cmnd)
 * This inserts the tape command cmnd at the head of
 * the tape, moving all of the other commands back one slot.
 *
 * Arguments: cmnd, tthe integer value of the new command
 * Returns: ERROR/OK
 * Requirements: conservation of momentum.
 * ------------------------------------------------------*/
insert_menu_cmnd_immediate(cmnd)
enum tape_cmnd cmnd;
    {
    if (((tail+1)% SIZE_MENU) == head) return (ERROR);
    head = (head-1+SIZE_MENU)%SIZE_MENU;
    menu[head] = (int)cmnd;
    return(OK);
    }

/* ------------------------------------------------------
 * add_menu_cmnd(cmnd)
```

```
 * This places cmnd at the tail of the command tape.
 *
 * Arguments: cmnd, the integer value of the new command.
 * Returns: ERROR/OK
 * Requirements: none.
 * ------------------------------------------------------*/
add_menu_cmnd(cmnd)
enum tape_cmnd cmnd;
    {
    if (((tail+1)% SIZE_MENU) == head) return (ERROR);
    menu[tail] = (int)cmnd;
    if (++tail == SIZE_MENU) tail = 0;
    return (OK);
    }

/* ------------------------------------------------------
 * increment_menu_ptr()
 * This command increments "head" to the next command.
 * The option of not doing this for any tape command
 * exists because as Fei-Yue describred the PNT, using
 * a tape command doesn't necessarally increment you to
 * the next command.
 *
 * Arguments:none
 * Returns:Error/ok
 * Requirements:none
 * ------------------------------------------------------*/
increment_menu_ptr()
    {
    if (tail == head) return (ERROR);
    menu[head] = (int)Empty;
    if (++head == SIZE_MENU) head=0;
    return (OK);
    }
```

A.6 petLib.c

This file contains most of the routines which are used in the manipulation of the Petri net. (eg. all the routines for firing the transitions...)

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: petLib.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**     main.c.
**     This file contains routines used to manipulate the
**     Petri nets (eg, fire transitions, check for enabeled
**     transitions)
**
** Modification History:
**
```

```
*/


#include <varargs.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <stdio.h>
#include "pet.h"
#include "dumb_dec.h"

extern struct place_ptr place_ptr_array[];
extern struct transition world[];
extern int graphics;
extern char title[];

/* ************************************************************
 * A routine used in timing.
 * the input "old" is a boolean which is true if the
 * function has been called before, and false if it has not.
 *
 * the value returned is the number of milliseconds since
 * this routine was last called.
 *
 * Arguments: old: a boolean.
 * Returns: the integer number of milliseconds passed since
 *    the routine was last called.
 * Requirements: timeb.h
 * ************************************************************/
int    time_diff(old)
int old;
{
    static time_t    seconds = 0;
    static unsigned short    millisec = 0;
    struct timeb    space;
    int ret_value;

    ftime(&space);
    if (old){
ret_value = 1000*(space.time-seconds) +
     (space.millitm-millisec);
seconds = space.time;
millisec = space.millitm;
```

```
return(ret_value);
}
    else    {
seconds = space.time;
millisec = space.millitm;
return(-1);
}
}

/* ********************************************************
 * This routineis called repeatedly to save the time
 * difference between the last time time_diff was called.
 *
 * Arguments: none
 * Returns: 1
 * Requirements: timeb.h
 * ********************************************************/
int time_holder_1[1000];
int rec_time1()
{
    static int arr_counter=0;
    static int first_call = TRUE;

    if (first_call){
 first_call = FALSE;
time_diff(FALSE);
}
    else time_holder_1[arr_counter++] = time_diff(TRUE);
    return (1);
}

/* ********************************************************
 * This routineis called repeatedly to save the time
 * difference between the last time time_diff was called.
 *
 * Arguments: none
 * Returns: 1
 * Requirements: timeb.h
 * ********************************************************/
int time_holder_2[1000];
int rec_time2()
{
    static int arr_counter=0;
```

```
    time_holder_2[arr_counter++] = time_diff(TRUE);
    return (1);
}

/* ********************************************************
 * This routine prints out the time_diff numbers to a file
 * then exits the program.
 *
 * Arguments: none
 * Returns: none
 * Requirements:none
 * ********************************************************/
void exit_program()
{
    int i;
    FILE *fp;

    fp = fopen("out1", "w");
    for (i=0;i<MAX_NUM_CYCLES/2;i++)
fprintf(fp, "%d \n",time_holder_1[i]);
    fclose(fp);
    fp = fopen("out2", "w");
    for (i=0;i<MAX_NUM_CYCLES/2;i++)
fprintf(fp, "%d \n",time_holder_2[i]);
    fclose(fp);
    printf("finish\n");
    exit(1);
}

/* ********************************************************
 * this function sends a message over the specified socket.
 * the message is the characters specified in the
 *  definition of PLAC (currently"pla" followed by the
 * integer array_num.  Upon reception the reader decodes
 * PLAC to realize that the following integer is the place
 * in an array that must be incremented.
 *
 * Arguments: Socknum, the integer number of the socket
 *    the message is to be transmitted over.
 *        array_num: the number of the place in the
 *    destination net.
 *        token_val: the value of the token to be sent.
```

```
 * Returns: 1
 * Requirements: socknum is an opened socket.
 * ********************************************************/
int sock_send(socknum, array_num,token_val)
int socknum, array_num,token_val;
{
int    buf[9 + (sizeof JUNK)/sizeof(int)];


    buf[0] = (int) PLAC;
    buf[1] = array_num;
    buf[2] = token_val;
    buf[3] = (int) DATA;
    buf[4] = sizeof JUNK;
    strcpy (&buf[5], JUNK);
    if (write (socknum, (char *) buf, 5*sizeof(int)+ sizeof JUNK)
    < 0)
perror("client:write:");
    return(1);
}

/* ************************************************************
 * This function should be called with a call like:
 * send_tape(socket_number, command_num, tape_cmnd1,
 *            tape_cmnd2, tape_cmnd3, NULL);
 * to send those three tape commands over the socket socket_number
 *
 * For some clarification see varargs(3), which will
 * clarify the variable number of arguments...
 *
 * Arguments: socket_number is the number of the socket.
 *        command_num is one of the enum type
 *    tape commands described in pet.h
 *        tape_cmnd.. are one of the enum type
 *     tape commands described in pet.h.
 * Returns: nothing
 * Requirements: var_args.h, socket_number is an open socket.
 * ********************************************************/
void  send_tape(va_alist)
va_dcl
{
va_list pointer;
    int buf[SIZE_MENU+9 + (sizeof JUNK)/sizeof(int)];
```

```
    int  sock_num, tape_cmnd, command_count=0,command_num;

va_start(pointer);
sock_num = va_arg(pointer,int);
command_num = va_arg(pointer,int);
while ((tape_cmnd=va_arg(pointer, int )) != NULL)
{
buf[1+ ++command_count] = tape_cmnd;
}
    buf[0] = command_num;
    buf[1] = command_count;
    if (command_count > SIZE_MENU) printf("to many commands,
 sending anyway,but fix this, or you'll get
  a segmentation fault next time.... \n");
    buf[command_count + 2] = (int) DATA;
    buf[command_count + 3] = sizeof JUNK;
    strcpy (&buf[command_count + 4], JUNK);

    if (write (sock_num, (char *) buf, (command_count +
    4)*sizeof(int)
+sizeof JUNK) < 0)
perror("client:write:");
va_end(pointer);
}


/* ************************************************************
 * A predicate which determines if a transition is enabled,
 * returning either true or false.
 *
 * Arguments: tran: a pointer to a transition.
 * Returns: FALSE/ value of the legitimate token.
 *    Note: this is a bug: one can't have 0 (== FALSE)
 *    as a token value.
 * Requirements: all the normal data structures.
 * ************************************************************/

int transition_enabled_p(trans)
struct transition *trans;
    {
    struct pre_pointer *next;
    int result = FALSE,carry_val,possible_value;
    struct enabelors *allowed;
```

```
    struct place_contains *item;


    if(trans->pre_places == NULL) return(FALSE);
    if (instruct_in_menu_p(trans))
{
for(allowed = trans->enabled_by; allowed != NULL;
    allowed = allowed->next_enabelor)
    {
    carry_val = TRUE;
    for (next = trans->pre_places; next != NULL;
 next = next->next_place)
{
result = FALSE;
if (next->place != NULL)
    {
    for (item = next->place->place;item !=
NULL;item= item->next_contents)
{
if ((*next->decoloration_routine)(next->place->place->object)!=
allowed->enabelor)
    {result = result || FALSE;}
else
    {result = TRUE;}
}
    }
carry_val = carry_val && result;
}
    if(carry_val) return(allowed->enabelor);
    if( allowed->next_enabelor == NULL) return (FALSE);
    }
/*
 * this next little loop covers the condition that there are no
 enabelors
 */
for (next = trans->pre_places; next != NULL;
     next = next->next_place)
    {
    if (next->place->place == NULL)
{return(FALSE);}
    else
{result =TRUE;
possible_value = next->place->place->object;
```

```
}
    }
if (result) return(possible_value);
else return (FALSE);
}
    else
{
return (FALSE);
}
}


/* ************************************************************
 * A routine which "fires" a transition.  Firing a transition
 * consists of:    Flashing the transition.
 *    removing one token (which must match the passed
 *         parameter "token") from each input place.
 *      adding one token to each output place.
 *      running the routine associated with the transition.
 *
 * Note that this routine should only be called if
 * transition_enabled_p(trans) returned a value of TRUE.
 *
 * Arguments: trans: a pointer to a transition.
 *        token: the value of the legitimate token
 *    to be removed.
 * Returns: the integer value of the routine associated with
 *    the transition
 * Requirements:
 * ************************************************************/
int fire_transition(trans,token)
int token;
struct transition *trans;
    {
    struct pre_pointer *next;
    struct post_pointer *nnext;
    int return_cond;

    if(graphics) flash_pltr(trans - world);

    next = trans->pre_places;
    while (next != NULL)
```

```
{
if (remove_token((next->place - &place_ptr_array[0])
    ,token,trans,
    next->decoloration_routine)== ERROR)
    {
    perror("fire_transition:remove_token:");
    exit(0);
    }
else {
    next= next->next_place;
    }
}
    nnext = trans->post_places;
    while (nnext != NULL)
{
if (!(nnext->remote)){
    add_token((nnext->place - &place_ptr_array[0]),
(*nnext->coloration_routine)(token));
     }
else if((*nnext->data_trans_routine)(nnext->remote,
    (nnext->place - &place_ptr_array[0])
    , (*nnext->coloration_routine)(token))
    != 1)
    {
    perror("fire_transition:add_token(remote):");
    exit(0);
    }
nnext= nnext->next_place;
}
    return_cond = (trans->routine)();
    if ((trans->menu_requirements != NULL) &&
    trans->consumes_menu)
if(increment_menu_ptr() == ERROR){
    printf("attempting to increment menu pointer with
    \n");
    printf("head == tail in %s Exiting.. \n",title);
    exit(1);
    }
    return (return_cond);
    }
```

```
/* ************************************************************
 * A whole bunch of dummy routines which may be attached
 * to the transitions.
 *
 * Arguments: none
 * Returns: Note: for these to match the rest of the code,
 *      all routines must return type int.
 * Requirements: see returns
 * ************************************************************/
int dummy(){    return(1);}
```

A.7 petri.c

This cryptically named file has the routines for calculating the offsets from the other arcs when computing the routing of the arcs.

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
```

```
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: petri.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains routines to calculate the offsets
**    an arc should have from the other arcs.
**
** Modification History:
**
*/

#include <stdio.h>
#include "pltr.h"
#include "draw.h"

#define True    1
#define False    0
#define None    (-1)


/* pltrtype    pltr[MaxPlTr];

arctype        arc[MaxArc];

int    npltr, narc; */

/* ------------------------------------------------------------
 *  This routine calculates the y offset within a row
 * for an arc. (eg, it might be in the 4th row, which
 * is a cm wide, and 6 mm from the top of it.
 *
 * Arguments: col1, col2, row, integers stateing the beginning
 *     and ending points of a segemnt of an arc.
 * Returns: integer: the offset value
```

```
 * Requirements:
 * ---------------------------------------------------------*/
xoff(col1, col2, row)
int    col1, col2, row;
{
    int    i, off;

    for(i=off=0; i < narc; off++)
for (i=0; i<narc; i++) {
    if (arc[i].used && arc[i].y1 == row &&
    overlap(col1, col2,
arc[i].x0, arc[i].x2) &&
    arc[i].off1 == off)
break;
    if (arc[i].used && arc[i].y3 == row &&
    overlap(col1, col2,
arc[i].x2, arc[i].x4) &&
    arc[i].off3 == off)
break;
}
    return off-1;
}

/* ---------------------------------------------------------
 *  This routine calculates the x offset within a column
 * for an arc. (eg, it might be in the 4th column, which
 * is a cm wide, and 6 mm from the left of it.
 *
 * Arguments: row1, row2, col, integers stateing the beginning
 *     and ending points of a segemnt of an arc.
 * Returns: integer: the offset value
 * Requirements:
 * ---------------------------------------------------------*/
yoff(row1, row2, col)
int    row1, row2, col;
{
    int    i, off;

    for(i=off=0; i < narc; off++)
for(i=0; i<narc; i++) {
    if (arc[i].used && arc[i].x0 == col &&
    overlap(row1, row2,
arc[i].y0, arc[i].y1) &&
```

```
    arc[i].off0 == off)
break;
    if (arc[i].used && arc[i].x2 == col &&
    overlap(row1, row2,
arc[i].y1, arc[i].y3) &&
    arc[i].off2 == off)
break;
    if (arc[i].used && arc[i].x4 == col &&
    overlap(row1, row2,
arc[i].y3, arc[i].y5) &&
    arc[i].off4 == off)
break;
}
    return off-1;
}

/* ------------------------------------------------------------
 * A trivial boolean
 *
 * Arguments: all integers.
 * Returns:true/false depending if a is between x and y or not
 * Requirements:
 * ------------------------------------------------------------*/
between(x, y, a)
{
    return ((y >= a && a >= x) || (y <= a && a <= x));
}

/* ------------------------------------------------------------
 *
 * Arguments:
 * Returns:
 * Requirements:
 * ------------------------------------------------------------*/
overlap(a, b, x, y)
int    a, b, x, y;
{
    return ( between(x, y, a) || between(x, y, b) ||
 (a==x && b==y) || (a==y && b==x) );
}
```

A.8 postn.c

This file contains the routine that actually calculates the routes for the arcs.

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File:postn.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains the routine that positions the arcs
**    based on their start and end points.
```

```
**
** Modification History:
**
*/
#include <stdio.h>
#include "pltr.h"


/* ------------------------------------------------------
 *
 * Arguments: i the number of the arc to be positioned.
 * Returns: nothing
 * Requirements:
 * ------------------------------------------------------*/

void arc_postn(i)
int    i;
{
    int    xfrom,  yfrom, xto, yto, from, to;


    from = arc[i].src;
    to   = arc[i].dest;

    xfrom = pltr[from].x;
    yfrom = pltr[from].y;

    xto = pltr[to].x;
    yto = pltr[to].y;

    arc[i].y0 = yfrom;
    arc[i].x0 = xfrom;
    arc[i].off0 = yoff(arc[i].y0, arc[i].y1, arc[i].x0);

    arc[i].y1 = yfrom + 1;

    if (xto < xfrom)
 arc[i].x2 = xto + 1;
    else if (xto > xfrom)
 arc[i].x2 = xto - 1;
    else if ((yfrom+2) == yto)
 arc[i].x2 = xto;
    else
```

```
arc[i].x2 = xto - 1;

    arc[i].off1 = xoff(arc[i].x0, arc[i].x2, arc[i].y1);

    arc[i].y3 = yto - 1;
    arc[i].off2 = yoff(arc[i].y1, arc[i].y3, arc[i].x2);

    arc[i].x4 = xto;
    arc[i].off3 = xoff(arc[i].x2, arc[i].x4, arc[i].y3);

    arc[i].y5 = yto;
    arc[i].off4 = yoff(arc[i].y3, arc[i].y5, arc[i].x4);

}
```

A.9 read_socket.c

This file contains the routine which reads the socket upon getting an interupt.

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
```

```
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: read_socket.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains a routine to read all of the sockets
**    available to a particular program, and deal with any
**    incomming data according to the established protocol.
**
** Modification History:
**
*/

#include <stdio.h>
#include <signal.h>
#include "pet.h"
#include "dumb_dec.h"

extern  int     should_pause,graphics;
extern int  socket_arr[];
int io_came;
extern char title[];


/* ------------------------------------------------
 * This routine sets a boolean which is used by main()
 * whenever anything comes in over the socket.
 *
 * Arguments: none
 * Returns: nothing
 * Requirements: the interupt must be enabeled (this is
 *    done in the sock_start() routines.
 * ------------------------------------------------*/
void io_interupt_handler(){
```

```
io_came = TRUE;
should_pause = FALSE;
}

/* ------------------------------------------------------------
 * This routine reads the all of the sockets until they are
 * empty.
 * The routine looks long, but it just does the same thing
 * over and over.  Basically:
 * i =0;
 * while(i < number of sockets)
 *    {
 *     data = read(socket[i])
 *    if(data == PLAC)
 *        then the message says to add a token to a
 *        place, deal with it.
 *    else if(data == ADD_TAPE_END)
 *         then the message says to add somthing to
 *        the end of the tape, deal with it.
 *    else if(data == ADD_TAPE_IMMEDIATE)
 *         then the message says to add somthing to
 *        the current position of the tape, deal with it.
 *    else increment to the next socket.
 *    }
 *
 * Arguments: number_of_sockets.  tells the routine how many to
 check.
 * Returns: Nothing
 * Requirements: An open socket for each number_of_sockets
 * ------------------------------------------------------------*/

void    io_handler(number_of_sockets){
int i=0 ,j,  array_num,type, command_number;
char data_byte;

io_came = FALSE;
while (i < number_of_sockets){
    if (read(socket_arr[i], &command_number, sizeof(int)) < 0){
    perror("reading stream message (first)");
    exit(1);
    }
    else if (command_number == (int) PLAC)
    {
```

```
    if (read(socket_arr[i], &array_num, sizeof(int)) < 0){
    perror("reading stream message(second)");
    exit(1);
    }
    if (read(socket_arr[i], &type, sizeof(int)) < 0){
perror("reading stream message(third)");
    exit(1);
    }
    add_token(array_num, type);
    should_pause = FALSE;
    i = 0;
    }
    else if (command_number == (int) AD_TAPE_END )
    {/* add stuff dealing with adding tape data...*/
     /* note that commands to add stuff to the tape are
of the form:
     AD_TAPE_END number_of_additions addition0
     addition1....
     */
    if (read(socket_arr[i], &array_num, sizeof(int)) < 0){
    perror("reading stream message(fourth)");
    exit(1);
    }
    if(array_num > SIZE_MENU)
    printf("Attempting to add to many tape
    commands \n");
    for(j=0;j<array_num; j++){
    if (read(socket_arr[i], &type, sizeof(int)) <
    0){
    perror("reading stream
    message(fifth)");
    exit(1);
    }
    if(add_menu_cmnd((enum tape_cmnd) type) ==
    ERROR)
    printf("tape add failed \n");
 if (graphics) draw_tape_box(current_tail_of_tape() -1);
    }
    i = 0;
    }
    else if (command_number == (int) AD_TAPE_IMMEDIATE )
    {       /* add stuff dealing with adding tape
    data...*/
```

```
    /* note that commands to add stuff to the tape
    are
       of the form:
    AD_TAPE_IMMEDIATE  number_of_additions
addition5 addition4 addition3....
Note that they are in reverse order because the
insertion process switches the order.
    */
    if (read(socket_arr[i], &array_num, sizeof(int)) < 0){
    perror("reading stream message(fourth)");
    exit(1);
    }
    if(array_num > SIZE_MENU)
    printf("Attempting to add to many tape
    commands \n");
    for(j=0;j<array_num; j++){
    if (read(socket_arr[i], &type, sizeof(int)) <
    0){
    perror("reading stream
    message(fifth)");
    exit(1);
    }
    if(insert_menu_cmnd_immediate((enum tape_cmnd)
    type)
     == ERROR)
    printf("tape add failed \n");
if (graphics)
draw_tape_box(current_head_of_tape());

    }
    i = 0;
    }
    else if (command_number == (int) DATA)
{
if (read(socket_arr[i], &array_num, sizeof(int)) < 0){
    perror("reading stream message (eigth)");
    exit(1);
    }
for(j=0;j<array_num;j++){
    if (read(socket_arr[i], &data_byte, sizeof(char)) <0){
perror("reading stream (ninth)");
exit(1);
}
```

```
    /* this is where a routine doing something with */
    /* the data would be.                           */
    }
i = 0;
    }
    else i++;
command_number = 0;
    }
should_pause = FALSE;
}
```

A.10 sock_connect.c

This file handles connecting to already existing sockets.

```
/*
**                    NOTICE OF COPYRIGHT
**           Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
```

```
**
** File: sock_connect.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains a routine to connect to the specified
**    pre-existing socket(s)
**
** Modification History:
**
*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/uio.h>
#include <errno.h>
#include <signal.h>
#include "pet.h"
#include "dumb_dec.h"

#define NUM_INFO_PER_SOCK 2

/* This program creates a client process
   on a socket by asking for a connection */

int socket_arr[LOCAL_SOCKS];
extern struct  place_ptr place_ptr_array[];

/* ------------------------------------------------------
 * This routine connects to the sockets specified in the
 * command line argument.  The socklets are marked as
 * Asynchronous, non-blocking sockets.
 *
 * Arguments: argv, argc.  argv is an array of character
 *    strings.  Since the command line for any module which
 *    uses sock_connect is
```

```
 *    <cmnd><location><number>[<location><number>] [-fast]
 *    the locations and numbers are read from that array
 *    and used by the routine.
 * Returns: Nothing
 * Requirements: This must be called after the sockets are
 *     opened.  (not much of a problem, as you don't have
 *    enough information to call this until the socket is
 *    opened.
 * ------------------------------------------------------*/
void sock_start(argc,argv)
int argc;
char *argv[];
{
    int sock_num[LOCAL_SOCKS],i;
    struct sockaddr_in server;
    struct hostent *hp, *gethostbyname();

if ((argc < NUM_INFO_PER_SOCK*LOCAL_SOCKS +1)|| (argc >
NUM_INFO_PER_SOCK*
LOCAL_SOCKS +2)){
    fprintf (stderr, "The number of local sockets (in pet.h) does
    not\n");
    fprintf (stderr, "match the number given implicitly in the
    command\n");
    fprintf (stderr, "Local socks = %d, you should give the host
    name \n");
    fprintf (stderr, "and socket number for each socket\n");
    exit(1);
    }

/* this is the line which tells the interupts where to go... */
signal (SIGIO, io_interupt_handler);

    for (i=0;i<LOCAL_SOCKS;i++){
sock_num[i] = atoi(argv[2+NUM_INFO_PER_SOCK*i]);
if ((socket_arr[i] = socket(AF_INET,SOCK_STREAM,0))<0){
    perror("client:open:"); exit(1);}

server.sin_family = AF_INET;
hp = gethostbyname(argv[NUM_INFO_PER_SOCK * i+1]);
if (hp == 0) {
    fprintf (stderr , "%s: unknown host",
argv[NUM_INFO_PER_SOCK * i+1]);
```

```
    exit (2);
    }
bcopy((char *)hp->h_addr,(char
*)&server.sin_addr,hp->h_length);
server.sin_port = htons(sock_num[i]);

if (connect(socket_arr[i],(struct sockaddr *)&server,
    sizeof server)<0){
    perror ("client:connect:");
    exit(1);
    }
    printf("opened socket to %s %d = %d \n ",
argv[NUM_INFO_PER_SOCK * i+1], sock_num[i],socket_arr[i]);
if (fcntl(socket_arr[i],F_SETOWN, getpid())<0){
perror("fcntl F_SETOWN, :");
exit(1);
    }
if (fcntl(socket_arr[i],F_SETFL, FASYNC|FNBIO)<0){
perror("fcntl F_SETFL, FNBIO|FASYNC");
exit(1);
    }
    }
}
```

A.11 sock_open.c

This routine opens up sockets for sock_connect to connect to.

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
```

```
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: sock_open.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains a routine to open to the specified
**    socket(s)
**
** Modification History:
**
*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include "pet.h"
#include "dumb_dec.h"

/* This program sets up a server (connection
   receiver) to listen for a message on a socket.
   When it receives a message, it displays the
```

```
   message on stdout, sends the same message back
   to the sender, and exits */

/*  This program is called by main2
 *  It sets up some sockets and writes their addresses out to  the
 screen
 */

extern    int    place[];
int socket_arr[MAIN_SOCKS];

/* ------------------------------------------------------
 * This routine opens MAIN_SOCKS sockets.
 * The socklets are marked as Asynchronous, non-blocking
 * sockets.
 *
 * Arguments: none. (argv and argc are ignored, they are
 *    there so this routine looks like sock_connect.
 * Returns: Nothing
 * Requirements: none
 * ------------------------------------------------------*/
void sock_start(argc, argv)
int argc;        /* these are to make this program */
char *argv[];        /* compatable with sock_start in
sock_connect.c */
{
    int ns[MAIN_SOCKS],i, sock_nums[MAIN_SOCKS],stringlen;
    int j,connections_need_to_be_made,
    need_to_connect_to[MAIN_SOCKS];
    struct    sockaddr_in sock_name[MAIN_SOCKS];

/* specify the name of the routine that handles SIGIO interupts */
    signal (SIGIO, io_interupt_handler);

/* Set the permission so that the SIGIO/SIGURG interupts can be
sent
 * set up listening sockets, and allow receipt of asynchronous I/O
 * signals
 */
    for (i=0;i<MAIN_SOCKS;i++){
if((ns[i] = socket(AF_INET,SOCK_STREAM, 0))<0){
 perror("server:socket:opening error:");
     exit(1);
```

```
}
    if (fcntl(ns[i],F_SETFL, FNBIO)<0){
perror("fcntl F_SETFL, FNBIO");
exit(1);
      }
    }

/* Name the socket using wildcards    */
    for (i=0;i<MAIN_SOCKS;i++){
    sock_name[i].sin_family = AF_INET;
    sock_name[i].sin_addr.s_addr = INADDR_ANY;
    sock_name[i].sin_port = 0;
    if(bind(ns[i],(struct sockaddr *)&sock_name[i],sizeof
    sock_name[i])==-1){
perror("server:bind");
exit(1);
    }

    stringlen = sizeof sock_name[i];
    if(getsockname(ns[i],(struct sockaddr *)&sock_name[i],
    &stringlen)<0){
perror("getting socket name:");
exit(1);
    }
    sock_nums[i] = ntohs(sock_name[i].sin_port);
    printf ("socket port #%d has opened \n",
    ntohs(sock_name[i].sin_port));
    }

    for (i=0;i<MAIN_SOCKS;i++)
    listen(ns[i],3);

    /* accept connection request */
 /* these should be marked as non-blocking, and go through, and
 attempt to
  * accept until all of the connections are accepted.  Note that
  the hassle
  * of dealing with non-blocking sockets is so that we don't have
  to connect
  * in order.
  */

 for (i=0;i<MAIN_SOCKS;i++)
```

```
    need_to_connect_to[i] = TRUE;
connections_need_to_be_made = TRUE;
while (connections_need_to_be_made){
    for (i=0;i<MAIN_SOCKS;i++){
    if(need_to_connect_to[i]){
socket_arr[i] = accept(ns[i],(struct sockaddr *)0, (int
*)0);
    if ((socket_arr[i] < 0)&& (errno != EWOULDBLOCK))
perror("server:accept");
    else if (errno != EWOULDBLOCK){
 printf("connection opened\n");
need_to_connect_to[i] = FALSE;
if (fcntl(socket_arr[i],F_SETOWN, getpid())<0){
    perror("fcntl F_SETOWN, :");
    exit(1);
   }
if (fcntl(socket_arr[i],F_SETFL, FASYNC|FNBIO)<0){
    perror("fcntl F_SETFL, FASYNC");
    exit(1);
    }
connections_need_to_be_made = FALSE;
for(j=0;j<MAIN_SOCKS;j++)
     connections_need_to_be_made |=
     need_to_connect_to[j];
    }
    }
    }
}
}
```

A.12 transform.c

This routine translates from the default data structure to one which the drawing routines use.

```
/*
**                      NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
```

```
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: transform.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains routines to transform the the data
**    to the pltr data structures.  This is needed because
**    drawing routines use the pltr data structures, and
**    the firing routines use the other data structures.
**    This should probably be modified, but "if it ain't
**    broke don't fix it".
**
** Modification History:
**
*/
#include "pltr.h"
#include "pet.h"
#include "dumb_dec.h"
```

```
#define MAX_PLACES 70 /* note, this number is likely to be */
      /* one of the things giving you      */
      /* trouble when you remake the nets  */

extern struct place_ptr place_ptr_array[];
extern int num_transitions;

/* ------------------------------------------------------------
 * fill_pltr_arc_map(world)
 * This routine takes the set of data structures defined in
 * world and fills the pltr data structures, so that the
 * graphics package can be used.
 *
 * quick pseudocode:
 * for each transition:
 *   put that transition on the pltr list.
 *   for each preplace of that transition
 *     if that place has not yet been put into the pltr list
 *        put it in the list
 *        mark the connections betweeen the place and transition.
 *        put the arc between them on the arc list
 *     otherwise
 *        mark the connections between the place and the
 transition.
 *        put the arc between them on the arc list
 *   for each postplace of that transition
 *     if that place has not yet been put into the pltr list
 *        put it in the list
 *        mark the connections betweeen the place and transition.
 *        put the arc between them on the arc list
 *     otherwise
 *        mark the connections between the place and the
 transition.
 *        put the arc between them on the arc list
 *
 *Note, the pltr list is a list of all the places and
 transitions.
 *
 * Arguments: world: an array of all the transitions.
 * Returns: nothing
 * Requirements: Note that this is called only after the other
 *    data structure is completed.
 * ------------------------------------------------------------*/
```

```
void fill_pltr_arc_map(world)
struct transition world[];
{
    int arc_counter, i,place_num;
    struct pre_pointer *next;
    struct post_pointer *nnext;

    arc_counter = 0;
    for (i=0;i<num_transitions;i++)        /* for each transition
    */
{
if ((world[i].pre_places != NULL)||
    (world[i].post_places != NULL))
    mark_tran(i,world[i].x_loc, world[i].y_loc);
next = world[i].pre_places;
while (next != NULL){
    if((place_num = already_listed_p(next->place)) <0)
{
place_num = -1*place_num + num_transitions;
fix_arc_pointer(i,place_num, arc_counter,TRUE);
}
    else
{
place_num = place_num + num_transitions;
mark_place(place_num, arc_counter, TRUE,
    next->place->x_loc, next->place->y_loc);
mark_arc(i,place_num, arc_counter,TRUE);
}
    arc_counter++;
    next = next->next_place;
    }
nnext = world[i].post_places;
while (nnext != NULL){
    if((place_num = already_listed_p(nnext->place)) <0)
{
place_num = -1*place_num + num_transitions;
fix_arc_pointer(i,place_num, arc_counter,FALSE);
}
    else
{
place_num = place_num + num_transitions;
mark_place(place_num, arc_counter, FALSE,
    nnext->place->x_loc, nnext->place->y_loc);
```

```
mark_arc(i,place_num, arc_counter,FALSE);
}
    arc_counter++;
    nnext = nnext->next_place;
    }
    }
}


/* ---------------------------------------------------------
 *  checks to see if the place has already been added to the
 * database, if it has then this returns -1* it's location.
 * Otherwise it returns its new location.
 *
 * Arguments: canidate_place.  A pointer to the place being
 checked.
 * Returns: A number (place in in the pltr array) for that
 *    place.  (if the number is negitive the place has
 *    already been located.
 * Requirements:
 * ---------------------------------------------------------*/
int already_listed_p(canidate_place)
struct place_ptr *canidate_place;
{
static int place_markers[MAX_PLACES] = {0};
int i;

i = canidate_place - &(place_ptr_array[0]);
if (place_markers[i]) return (-1 * i);
else place_markers[i] = TRUE;
return (i);
}


/* ---------------------------------------------------------
 * this deals with changing all of the data if a new arc
 * is to be added to a place which was already declared.
 * Specifically, a place in the arc array must be set
 * aside, and the linkled lists of arcs in the pltr lists
 * must have the new arc added.
 *
 * Arguments: tran_num, place_num, arc_num: the number in
 *    their respective arrays of the transition, place,
```

```
 *    and arc.
 *        pre: a boolean, true if the place is a
 *    pre-place to the transition.
 * Returns: nothing
 * Requirements:
 * ------------------------------------------------------*/
void fix_arc_pointer(tran_num, place_num, arc_num, pre)
int tran_num, place_num, arc_num, pre;
    /* note pre is a boolean indicating*/
    /* if the place is a pre-place or a*/
    /* post-place                */
{
int base_arc;

    arc[arc_num].srcnext = NullArc;
    arc[arc_num].destnext = NullArc;
    if (pre){
if (pltr[place_num].out == NullArc){
    pltr[place_num].out = arc_num;
    arc[arc_num].src = place_num;
    arc[arc_num].dest = tran_num;
    }
else{
    base_arc = pltr[place_num].out;
    while (arc[base_arc].srcnext != NullArc)
 { base_arc = arc[base_arc].srcnext;}
    arc[base_arc].srcnext = arc_num;
    arc[arc_num].src = place_num;
    arc[arc_num].dest = tran_num;
    }
fix_tran_input_arcs(tran_num, arc_num);
    }

    else{
if (pltr[place_num].in == NullArc){
    pltr[place_num].in = arc_num;
    arc[arc_num].dest = place_num;
    arc[arc_num].src = tran_num;
    }
else{
    base_arc = pltr[place_num].in;
    while (arc[base_arc].destnext != NullArc)
  { base_arc = arc[base_arc].destnext;}
```

```
    arc[base_arc].destnext = arc_num;
    arc[arc_num].dest = place_num;
    arc[arc_num].src = tran_num;
    }
fix_tran_output_arcs(tran_num, arc_num);
}
    arc[arc_num].used = TRUE;
    arc[arc_num].srcnext = NullArc;
    arc[arc_num].destnext = NullArc;
}

/* ---------------------------------------------------------
 * this routine fixes the pointers associated with the
 * transitions which have arcs leading into them.
 * (that is, this routine is called when we're adding the
 * arc arc_num to tran_num's list of arcs which input to
 * it.
 *
 * Arguments: tran_num, arc_num, the array number of the
 *    transition and arc.
 * Returns: nothing
 * Requirements:
 * ---------------------------------------------------------*/
void fix_tran_input_arcs(tran_num, arc_num)
int tran_num, arc_num;
{
int base_arc;
if (pltr[tran_num].in == NullArc){
    pltr[tran_num].in = arc_num;}
else{
    base_arc = pltr[tran_num].in;
    while (arc[base_arc].destnext != NullArc)
  base_arc = arc[base_arc].destnext;
    arc[base_arc].destnext = arc_num;
    }
}


/* ---------------------------------------------------------
 * this routine fixes the pointers associated with the
 * transitions which have arcs leading out of them.
 * (that is, this routine is called when we're adding the
 * arc arc_num to tran_num's list of arcs which output
```

```
 * from it.
 *
 * Arguments: tran_num, arc_num, the array number of the
 *    transition and arc.
 * Returns: nothing
 * Requirements:
 * ---------------------------------------------------------*/
void fix_tran_output_arcs(tran_num, arc_num)
int tran_num, arc_num;
{
int base_arc;
if (pltr[tran_num].out == NullArc){
    pltr[tran_num].out = arc_num;}
else{
    base_arc = pltr[tran_num].out;
    while (arc[base_arc].srcnext != NullArc)
 { base_arc = arc[base_arc].srcnext;}
    arc[base_arc].srcnext = arc_num;
    }
}

/* ---------------------------------------------------------
 * this routine records a place in the data structure the
 * first time it is encountered.
 *
 * Arguments: (sigh, isn't this obvious by now?)
 *    place_num, arc_num: the array number of the relevent things
 *    x_loc, y_loc, the desired screen locations.
 *    pre_place: boolean, true if pre-place relitive to
 *    the current transition when this place was "discovered"
 * Returns: Nothing, nada, zip.
 * Requirements:
 *---------------------------------------------------------*/
void mark_place(place_num, arc_num, pre_place,x_loc, y_loc)
int place_num, arc_num, pre_place;  /* note pre is a boolean */
int x_loc, y_loc;              /*indicating if the place is a pre-*/
     /*place or a post-place       */
{
    if (pre_place)    {pltr[place_num].out = arc_num;
     pltr[place_num].in = NullArc;}
    else          {pltr[place_num].in = arc_num;
     pltr[place_num].out = NullArc;}
    pltr[place_num].place = TRUE;
```

```
    pltr[place_num].used = TRUE;
    pltr[place_num].type = PlState;
    pltr[place_num].tok = 0;
    pltr[place_num].x = x_loc;
    pltr[place_num].y = y_loc;
}

/* ------------------------------------------------------
 * This fills the data structure for an arc associated with
 * a place the first time that that place is encountered...
 *
 * Arguments: tran_num, arc_num, place_num:
 *    the array locations of the relevent objects.
 *        pre: boolean, true if place is input to transition
 * Returns: nothing.
 * Requirements:
 * ------------------------------------------------------*/
void mark_arc(tran_num, place_num, arc_num, pre)
int tran_num, place_num, arc_num, pre;
    /* note pre is a boolean indicating*/
    /* if the place is a pre-place or a*/
    /* post-place                */
{
    arc[arc_num].used = TRUE;
    arc[arc_num].srcnext = NullArc;
    arc[arc_num].destnext = NullArc;
    if (pre){
arc[arc_num].src = place_num;
arc[arc_num].dest = tran_num;
fix_tran_input_arcs(tran_num, arc_num);
}
    else{
arc[arc_num].dest = place_num;
arc[arc_num].src = tran_num;
fix_tran_output_arcs(tran_num, arc_num);
}
}

/* ------------------------------------------------
 * this routine initializes a transition
 *
 * Arguments: tran_num: the number of the transition?
 *        x_loc, y_loc: location of the  transition
```

```
 * Returns:nothing
 * Requirements:
 * ----------------------------------------------------*/
void mark_tran(tran_num,x_loc, y_loc)
int tran_num,x_loc, y_loc;
{
    pltr[tran_num].place = FALSE;
    pltr[tran_num].used = TRUE;
    pltr[tran_num].type = PlEvent;
    pltr[tran_num].tok = 0; /* fix this!!!!!! */
    pltr[tran_num].x = x_loc;
    pltr[tran_num].y = y_loc;
    pltr[tran_num].in = NullArc;
    pltr[tran_num].out = NullArc;
}

/* ------------------------------------------------------
 * removes a token from the pltr representation of place
 * place_num.
 * note the relitve lengths of documentation and code.
 *
 * Arguments: place number
 * Returns:nothing.
 * Requirements:
 * ----------------------------------------------------*/
void remove_tok_in_pltr_rep(place_num)
int place_num;
{
pltr[place_num + num_transitions].tok--;
}


/* ------------------------------------------------------
 * adds a token to the pltr representation of place
 * place_num.
 * note the relitve lengths of documentation and code.
 *
 * Arguments: place number
 * Returns:nothing.
 * Requirements:
 * ----------------------------------------------------*/
void add_tok_in_pltr_rep(place_num)
int place_num;
```

```
{
pltr[place_num + num_transitions].tok++;
}
```

A.13 window_manager.c

This routine opens and moniters the window.

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: window_manager
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
```

```
** Purpose: The purpose of the package can be found in the file
**     main.c.
**     This file contains a routine which moniters the window,
**     and updates it when needed.  It also sends signals to
**     it's child (the main routine) when the user gives
**     instructions it's to senile to handle.
**     This file also contains the routine which initially
**     creates the window.
**     As a final ingredient the file contains spam sauteed
**     in the finest grease-ridden-carp grease, and due to
**     a special deal with Dahli-lama Export/Import, 2
**     drams of yak hair.
**
** Modification History:
**
*/
#include "dumb_dec.h"
#include <signal.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

#define TRUE 1
#define FALSE 0

extern char title[];
char fontname[]={"6x10"};

    Display *display;
    Window net_window;
    GC gc;
    unsigned long foreground, background;

    XEvent event;
    Font font;
    KeySym key;
    XSizeHints hint;
    int screen;
    char text[10];

/* ------------------------------------------------------------
 * This routine creates the window.
 *
 *
```

```
 * Arguments: none
 * Returns:none
 * Requirements: X11 library?
 * ------------------------------------------------------*/
void init_window(){
    display = XOpenDisplay("");
    screen = DefaultScreen(display);
    background = WhitePixel (display, screen);
    foreground = BlackPixel(display , screen);
    font = XLoadFont(display,fontname);

    hint.x =0;      hint.y= 0;
    hint.width=350;    hint.height = 250;
    hint.flags = PPosition |PSize;

/* create window*/
    net_window = XCreateSimpleWindow (display,
    DefaultRootWindow(display),
hint.x, hint.y, hint.width, hint.height, 5,
foreground, background);

    XSetStandardProperties (display, net_window, title, title,
    None,
0,0, &hint);

/* GC initialization & creation */

    gc = XCreateGC(display, net_window, 0,0);
    XSetFont(display,gc,font);
    XSetBackground(display, gc, background);
    XSetForeground(display, gc, foreground);

/* input event selection */
    XSelectInput (display, net_window, KeyPressMask|
    ExposureMask);
}



/* ------------------------------------------------------
 * this routine is a nice little infinite loop which
 * waits for the user to move the window, or type something
 * into it, or someone to put another window over it,
```

```
 * or anything like that.
 *
 * The routine sends messages to it's child by using the
 * kill command (and the child never sends messages back,
 * isn't that just typical?) which is why the child's pid is
 * passed to the routine.
 *
 * Arguments: pid. the pid of the process which needs to be
 *    notified to refresh the screen, or has to be killed
 *     when the user tells the program to end.
 * Returns:Never.
 * Requirements:X11 stuff, and the child needs to know what to
 *     do with a SIGUSR1 signal.
 * ----------------------------------------------------------*/
event_reading_loop(pid)
int pid;
    {
int not_done = TRUE,i;

while(not_done)
{
    XWindowEvent (display, net_window,
(long)KeyPressMask|ExposureMask, &event );

    switch(event.type)
{
    case Expose:
while(XCheckWindowEvent (display, net_window,
    (long)ExposureMask, &event ))
{/* empty all the expose events from the buffer..*/}
/* tell child to redraw screen */
    kill(pid,SIGUSR1);
break;

    case MappingNotify:
XRefreshKeyboardMapping (&event);
break;

/* process keyboard input */
    case KeyPress:
    i = XLookupString(&event, text, 10, &key, 0);
/* quit */
if (i==1 && text[0]== 'q') not_done = FALSE;
```

```
/* tell child to redraw screen */
if (i==1 && text[0]== 'r') kill(pid,SIGUSR1);
break;

    default:
printf("we fell through the case statement %s\n",title);
break;
}
    }
}
```

APPENDIX B
NET DESCRIPTIONS

B.1 setup_disp.c

This is the file describing the Dispatcher.

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: setup_disp.c
**
** Written By: Michael Mittmann
```

```
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains the net definition statements defining
**    the dispatcher.
**
** Modification History:
**
*/

#include <varargs.h>
#include "pet.h"
#include "pet2.h"
#include "dumb_dec.h"

#define NUM_TRANS 33
char title[] = {"Dispatcher"};

int number_of_sockets = MAIN_SOCKS;
int calibrate_menu(), look_menu(), grasp_menu(), release_menu(),
find_menu();
int continue_vision(), vision_task(), motion_task(), calr_menu(),
move_menu();
int approach_menu();

extern int rec_time1(), rec_time2();
extern int socket_arr[];
int menu[SIZE_MENU];
int num_transitions = NUM_TRANS;
struct place_ptr place_ptr_array[70];
struct transition world[NUM_TRANS];

/* ------------------------------------------------------------
 * make_net()
 * this routine just defines a net identical to the one described
 * in the GreatSPN1.5 file dispatcher_w_all_connect.
 *
 * Arguments:none
 * Returns:nothing
 * Requirements: just about all of the routines linked to it ;-)
 * ------------------------------------------------------------*/
```

```
make_net()
{
initialize_transition(calibrate_menu,TRAN world[0],4,12);
add_pre_list(&(world[0].pre_places), &place_ptr_array[30],
&place_ptr_array[42],
    NULL);
add_post_list(&(world[0].post_places), &place_ptr_array[32],0,
    &place_ptr_array[16],socket_arr[0],NULL);

initialize_transition(calibrate_menu,TRAN world[1],6,12);
add_pre_list(&(world[1].pre_places), &place_ptr_array[30],
&place_ptr_array[45],
     &place_ptr_array[39],NULL);
add_post_list(&(world[1].post_places), &place_ptr_array[34],0,
    &place_ptr_array[16],socket_arr[0],NULL);

initialize_transition(vision_task,TRAN world[2],8,12);
add_pre_list(&(world[2].pre_places), &place_ptr_array[30],
&place_ptr_array[44],
     &place_ptr_array[38],NULL);
add_post_list(&(world[2].post_places), &place_ptr_array[35],0,
    &place_ptr_array[16],socket_arr[0],NULL);

initialize_transition(vision_task,TRAN world[3],10,12);
add_pre_list(&(world[3].pre_places), &place_ptr_array[30],
&place_ptr_array[33],
    NULL);
add_post_list(&(world[3].post_places), &place_ptr_array[36],0,
    &place_ptr_array[16],socket_arr[0],NULL);

initialize_transition(vision_task,TRAN world[4],12,12);
add_pre_list(&(world[4].pre_places), &place_ptr_array[30],
&place_ptr_array[38],
     &place_ptr_array[45],NULL);
add_post_list(&(world[4].post_places), &place_ptr_array[37],0,
    &place_ptr_array[16],socket_arr[0],NULL);

initialize_transition(rec_time2,TRAN world[5],0,0);

initialize_transition(rec_time2,TRAN world[6],4,22);
add_pre_list(&(world[6].pre_places), &place_ptr_array[31],
```

```
&place_ptr_array[32],
    NULL);
add_post_list(&(world[6].post_places), &place_ptr_array[33],0,
    &place_ptr_array[20],socket_arr[0],NULL);

initialize_transition(rec_time2,TRAN world[7],6,22);
add_pre_list(&(world[7].pre_places), &place_ptr_array[31],
&place_ptr_array[34],
    NULL);
add_post_list(&(world[7].post_places), &place_ptr_array[33],0,
    &place_ptr_array[20],socket_arr[0],NULL);

initialize_transition(rec_time2,TRAN world[8],8,22);
add_pre_list(&(world[8].pre_places), &place_ptr_array[31],
&place_ptr_array[35],
    NULL);
add_post_list(&(world[8].post_places), &place_ptr_array[33],0,
    &place_ptr_array[20],socket_arr[0],NULL);

initialize_transition(rec_time2,TRAN world[9],10,22);
add_pre_list(&(world[9].pre_places), &place_ptr_array[31],
&place_ptr_array[36],
    NULL);
add_post_list(&(world[9].post_places), &place_ptr_array[33],0,
    &place_ptr_array[20],socket_arr[0],NULL);

initialize_transition(rec_time2,TRAN world[10],12,22);
add_pre_list(&(world[10].pre_places),&place_ptr_array[31],
&place_ptr_array[37],
    NULL);
add_post_list(&(world[10].post_places), &place_ptr_array[33],0,
    &place_ptr_array[20],socket_arr[0],NULL);

initialize_transition(motion_task ,TRAN world[11],22,2);
add_pre_list(&(world[11].pre_places),&place_ptr_array[40],
&place_ptr_array[46],
    NULL);
add_post_list(&(world[11].post_places), &place_ptr_array[41],0,
    &place_ptr_array[5],socket_arr[1],NULL);

initialize_transition(rec_time2,TRAN world[12],22,6);
add_pre_list(&(world[12].pre_places),&place_ptr_array[41],
&place_ptr_array[48],
```

```
    NULL);
add_post_list(&(world[12].post_places), &place_ptr_array[42],0,
    &place_ptr_array[9],socket_arr[1],NULL);

initialize_transition(motion_task ,TRAN world[13],22,10);
add_pre_list(&(world[13].pre_places),&place_ptr_array[42],
&place_ptr_array[46],
    NULL);
add_post_list(&(world[13].post_places), &place_ptr_array[43],0,
    &place_ptr_array[5],socket_arr[1],NULL);

initialize_transition(rec_time2,TRAN world[14],22,14);
add_pre_list(&(world[14].pre_places),&place_ptr_array[43],
&place_ptr_array[48],
    NULL);
add_post_list(&(world[14].post_places), &place_ptr_array[39],0,
    &place_ptr_array[44],0,&place_ptr_array[9],socket_arr[1],NULL);

initialize_transition(motion_task, TRAN world[15],28,16);
add_pre_list(&(world[15].pre_places),&place_ptr_array[59],
&place_ptr_array[46],
    NULL);
add_post_list(&(world[15].post_places), &place_ptr_array[50],0,
    &place_ptr_array[5],socket_arr[1],NULL);

initialize_transition(motion_task,TRAN world[16],28,18);
add_pre_list(&(world[16].pre_places),&place_ptr_array[44],
&place_ptr_array[46],
    NULL);
add_post_list(&(world[16].post_places), &place_ptr_array[51],0,
    &place_ptr_array[5],socket_arr[1],NULL);

initialize_transition(motion_task ,TRAN world[17],28,20);
add_pre_list(&(world[17].pre_places),&place_ptr_array[33],
&place_ptr_array[46],
    NULL);
add_post_list(&(world[17].post_places), &place_ptr_array[52],0,
    &place_ptr_array[5],socket_arr[1],NULL);

initialize_transition(motion_task,TRAN world[18],28,24);
add_pre_list(&(world[18].pre_places),&place_ptr_array[45],
&place_ptr_array[46],
    NULL);
```

```
add_post_list(&(world[18].post_places), &place_ptr_array[53],0,
    &place_ptr_array[5],socket_arr[1],NULL);

initialize_transition(rec_time2,TRAN world[19],32,20);
add_pre_list(&(world[19].pre_places),&place_ptr_array[50],
&place_ptr_array[48],
    NULL);
add_post_list(&(world[19].post_places), &place_ptr_array[44],0,
    &place_ptr_array[9],socket_arr[1],NULL);

initialize_transition(rec_time2,TRAN world[20],32,22);
add_pre_list(&(world[20].pre_places),&place_ptr_array[51],
&place_ptr_array[48],
    NULL);
add_post_list(&(world[20].post_places), &place_ptr_array[44],0,
    &place_ptr_array[9],socket_arr[1],NULL);

initialize_transition(rec_time2,TRAN world[21],32,24);
add_pre_list(&(world[21].pre_places),&place_ptr_array[52],
&place_ptr_array[48],
    NULL);
add_post_list(&(world[21].post_places), &place_ptr_array[44],0,
    &place_ptr_array[38],0,&place_ptr_array[9],socket_arr[1],NULL);

initialize_transition(rec_time2,TRAN world[22],32,28);
add_pre_list(&(world[22].pre_places),&place_ptr_array[53],
&place_ptr_array[48],
    NULL);
add_post_list(&(world[22].post_places), &place_ptr_array[45],0,
    &place_ptr_array[9],socket_arr[1],NULL);

initialize_transition(grasp_menu,TRAN world[23],14,28);
add_pre_list(&(world[23].pre_places),&place_ptr_array[44],
&place_ptr_array[54],
    NULL);
add_post_list(&(world[23].post_places), &place_ptr_array[56],0,
    &place_ptr_array[0],socket_arr[2],NULL);

initialize_transition(release_menu,TRAN world[24],18,28);
add_pre_list(&(world[24].pre_places),&place_ptr_array[45],
&place_ptr_array[54],
    NULL);
add_post_list(&(world[24].post_places), &place_ptr_array[57],0,
```

```
    &place_ptr_array[0],socket_arr[2],NULL);

initialize_transition(grasp_menu,TRAN world[25],22,28);
add_pre_list(&(world[25].pre_places),&place_ptr_array[59],
&place_ptr_array[54],
    NULL);
add_post_list(&(world[25].post_places), &place_ptr_array[58],0,
    &place_ptr_array[0],socket_arr[2],NULL);

initialize_transition(rec_time2,TRAN world[26],14,32);
add_pre_list(&(world[26].pre_places),&place_ptr_array[56],
&place_ptr_array[55],
    NULL);
add_post_list(&(world[26].post_places), &place_ptr_array[45],0,
    &place_ptr_array[4],socket_arr[2],NULL);

initialize_transition(rec_time2,TRAN world[27],18,32);
add_pre_list(&(world[27].pre_places),&place_ptr_array[57],
&place_ptr_array[55],
    NULL);
add_post_list(&(world[27].post_places), &place_ptr_array[59],0,
    &place_ptr_array[4],socket_arr[2],NULL);

initialize_transition(rec_time2,TRAN world[28],22,32);
add_pre_list(&(world[28].pre_places),&place_ptr_array[58],
&place_ptr_array[55],
    NULL);
add_post_list(&(world[28].post_places), &place_ptr_array[45],0,
    &place_ptr_array[4],socket_arr[2],NULL);

initialize_transition(continue_vision,TRAN world[29],2,12);
add_pre_list(&(world[29].pre_places),&place_ptr_array[30],
&place_ptr_array[62],
    NULL);
add_post_list(&(world[29].post_places), &place_ptr_array[61],0,
    &place_ptr_array[16],socket_arr[0],NULL);

initialize_transition(rec_time2,TRAN world[30],2,22);
add_pre_list(&(world[30].pre_places),&place_ptr_array[31],
&place_ptr_array[61],
    NULL);
add_post_list(&(world[30].post_places), &place_ptr_array[33],0,
    &place_ptr_array[20],socket_arr[0],NULL);
```

```
initialize_transition(motion_task, TRAN world[31],28,22);
add_pre_list(&(world[31].pre_places),&place_ptr_array[46],
&place_ptr_array[33],
    NULL);
add_post_list(&(world[31].post_places), &place_ptr_array[60],0,
    &place_ptr_array[5],socket_arr[1],NULL);

initialize_transition(rec_time2,TRAN world[32],32,26);
add_pre_list(&(world[32].pre_places),&place_ptr_array[48],
&place_ptr_array[60],
    NULL);
add_post_list(&(world[32].post_places), &place_ptr_array[62],0,
    &place_ptr_array[9],socket_arr[1],NULL);


declare_enab_tokens(&(world[11].menu_requirements), CalR, NULL);
declare_enab_tokens(&(world[13].menu_requirements), Move ,NULL);
declare_enab_tokens(&(world[15].menu_requirements), Move,
Approach,NULL);
declare_enab_tokens(&(world[16].menu_requirements), Move,
Approach,NULL);
declare_enab_tokens(&(world[17].menu_requirements), Move,
Approach,NULL);
declare_enab_tokens(&(world[18].menu_requirements), Move,
Approach,NULL);
declare_enab_tokens(&(world[0].menu_requirements), CalV,NULL);
declare_enab_tokens(&(world[1].menu_requirements), CalV, NULL);
declare_enab_tokens(&(world[2].menu_requirements), Look, Find
,NULL);
declare_enab_tokens(&(world[3].menu_requirements), Look, Find
,NULL);
declare_enab_tokens(&(world[4].menu_requirements), Look, Find
,NULL);
declare_enab_tokens(&(world[23].menu_requirements), Rel,NULL);
declare_enab_tokens(&(world[24].menu_requirements), Grip,NULL);
declare_enab_tokens(&(world[25].menu_requirements), Rel,NULL);
declare_enab_tokens(&(world[29].menu_requirements), Conti,NULL);
declare_enab_tokens(&(world[31].menu_requirements), Slave,NULL);

add_menu_cmnd(CalR);
add_menu_cmnd(CalV);
add_menu_cmnd(Move);
```

```
add_menu_cmnd(Rel);
add_menu_cmnd(Grip);
add_menu_cmnd(Move);
add_menu_cmnd(Approach);
add_menu_cmnd(Rel);
add_menu_cmnd(Look);
add_menu_cmnd(Find);
add_menu_cmnd(Move);
add_menu_cmnd(Find);
add_menu_cmnd(Move);
add_menu_cmnd(Move);
add_menu_cmnd(Approach);

place_place(30,0,10);
place_place(31,0,20);
place_place(32,4,18);
place_place(33,4,28);
place_place(34,6,18);
place_place(35,8,18);
place_place(36,10,18);
place_place(37,12,18);
place_place(38,16,10);
place_place(39,18,8);
place_place(40,22,0);
place_place(41,22,4);
place_place(42,22,8);
place_place(43,22,12);
place_place(44,22,18);
place_place(45,20,24);
place_place(46,26,0);
place_place(48,32,4);
place_place(50,30,18);
place_place(51,30,20);
place_place(52,30,22);
place_place(53,30,26);
place_place(54,10,26);
place_place(55,10,30);
place_place(56,14,30);
place_place(57,18,30);
place_place(58,22,30);
place_place(59,24,24);
place_place(60,30,24);
place_place(61,2,18);
```

```
place_place(62,2,30);
place_place(0, 28,30);
place_place(4, 28,34);
place_place(5, 30,4);
place_place(9, 36,8);
place_place(16, 0,16);
place_place(20, 0,24);
}
void initialize_marking(){
add_token(40,1);
add_token(40,1);
add_token(40,1);
add_token(46,1);
add_token(46,1);
add_token(30,1);
add_token(30,1);
add_token(54,1);
add_token(54,1);
}

int release_menu()
{
    rec_time1();
send_tape(socket_arr[2],AD_TAPE_END, GoPos, Retur, NULL);
    return(1);
}

int grasp_menu()
{
    rec_time1();
send_tape(socket_arr[2],AD_TAPE_END, Cross, GoPos,
Retur, NULL);
    return(1);
}

int calibrate_menu()
{
    rec_time1();
    send_tape(socket_arr[0],AD_TAPE_END, CalV, CalV, CalV, CalV,
    CalV, CalV, Retur, NULL);
    return(1);
}
```

```
int find_menu()
{
    send_tape(socket_arr[0],AD_TAPE_END, Find, Retur, NULL);
    return(1);
}


int look_menu()
{
    send_tape(socket_arr[0],AD_TAPE_END, Look, Retur,  NULL);
    return(1);
}

int vision_task()
    {
    rec_time1();
    if (get_menu_value() == (int)Find) return(find_menu());
    else return(look_menu());
    }

int motion_task()
    {
    rec_time1();
    if (get_menu_value() == (int) Move) return(move_menu());
    else if (get_menu_value() == (int) Approach)
    return(approach_menu());
    else if (get_menu_value() == (int) Slave) return(move_menu());
    else return(calr_menu());
    }

int calr_menu()
{
    send_tape(socket_arr[1],AD_TAPE_END, CalR, NULL);
    return(1);
}

int approach_menu()
{
    send_tape(socket_arr[1],AD_TAPE_END, Approach, NULL);
    return(1);
}

int move_menu()
```

```
{
    send_tape(socket_arr[1],AD_TAPE_END, Move, NULL);
    return(1);
}

int continue_vision()
{
    rec_time1();
    send_tape(socket_arr[0],AD_TAPE_IMMEDIATE, Conti, NULL);
    return(1);
}
```

B.2 setup_arm.c

This file describes the Petri net for the Motion Coordinator.

```
**
** Package: TokenPasser
**
** File: setup_arm.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains the statements needed to define a net
**    identical to the one in ~mittmann/nets/motion_full_io
**
** Modification History:
**
*/
#include <varargs.h>
#include "pet.h"
#include "pet2.h"
#include "dumb_dec.h"

#define NUM_TRANS 9

char title[] = {"Arm"};
extern int socket_arr[];
int number_of_sockets = LOCAL_SOCKS;
int menu[SIZE_MENU];
int num_transitions = NUM_TRANS;
struct place_ptr place_ptr_array[16];
struct transition world[NUM_TRANS];

/* ------------------------------------------------------------
 * make_net()
 * this routine just defines a net identical to the one described
 * in the GreatSPN1.5 file motion_full_io.
 *
 * Arguments:none
 * Returns:nothing
 * Requirements: just about all of the routines linked to it ;-)
 * ------------------------------------------------------------*/

make_net()
```

```
{
initialize_transition(dummy,TRAN world[0],2,4);
add_pre_list(&(world[0].pre_places), &place_ptr_array[5],
&place_ptr_array[6],
    NULL);
add_post_list(&(world[0].post_places), &place_ptr_array[7],0,
    NULL);

initialize_transition(dummy,TRAN world[1],2,8);
add_pre_list(&(world[1].pre_places), &place_ptr_array[7],
    NULL);
add_post_list(&(world[1].post_places), &place_ptr_array[8],0,
    NULL);

initialize_transition(dummy,TRAN world[2],6,8);
add_pre_list(&(world[2].pre_places),
&place_ptr_array[7],&place_ptr_array[9],
    NULL);
add_post_list(&(world[2].post_places), &place_ptr_array[6],0,
    &place_ptr_array[46],socket_arr[0],
    &place_ptr_array[48],
    socket_arr[0],NULL);

initialize_transition(dummy,TRAN world[3],6,10);
add_pre_list(&(world[3].pre_places), &place_ptr_array[9],
&place_ptr_array[10],
     &place_ptr_array[11], NULL);
add_post_list(&(world[3].post_places), &place_ptr_array[6],0,
    &place_ptr_array[46],socket_arr[0],
    &place_ptr_array[48],
    socket_arr[0],NULL);

initialize_transition(dummy,TRAN world[4],2,12);
add_pre_list(&(world[4].pre_places), &place_ptr_array[8],
    NULL);
add_post_list(&(world[4].post_places), &place_ptr_array[13],0,
     &place_ptr_array[10],0,NULL);

initialize_transition(dummy,TRAN world[5],2,16);
add_pre_list(&(world[5].pre_places), &place_ptr_array[12],
&place_ptr_array[13],
    NULL);
add_post_list(&(world[5].post_places), &place_ptr_array[14],0,
```

```
    NULL);

initialize_transition(dummy,TRAN world[6],8,18);
add_pre_list(&(world[6].pre_places), &place_ptr_array[15],
    NULL);
add_post_list(&(world[6].post_places), &place_ptr_array[11],0,
     &place_ptr_array[12], 0, NULL);

initialize_transition(dummy,TRAN world[7],4,20);
add_pre_list(&(world[7].pre_places), &place_ptr_array[15],
    NULL);
add_post_list(&(world[7].post_places), &place_ptr_array[14],0,
    NULL);

initialize_transition(dummy,TRAN world[8],4,24);
add_pre_list(&(world[8].pre_places), &place_ptr_array[14],
    NULL);
add_post_list(&(world[8].post_places), &place_ptr_array[15],0,
    NULL);

declare_enab_tokens(&(world[0].menu_requirements), Move,
Approach,CalR,NULL);


place_place(5, 20,2);
place_place(6, 2, 2);
place_place(7, 2 ,6);
place_place(8, 2 ,10);
place_place(9, 20,6);
place_place(10,8,14);
place_place(11,6,16);
place_place(12,4,14);
place_place(13,2,14);
place_place(14,2,22);
place_place(15,6,22);
place_place(46, 20,10);
place_place(48, 20,14);
}
void initialize_marking(){
add_token(6,1);
add_token(9,1);
add_token(12,1);
}
```

B.3 setup_grip.c

This file describes the Petri net for the Gripper Coordinator.

```
/*
**                    NOTICE OF COPYRIGHT
**          Copyright (C) Rensselaer Polytechnic Institute.
**                    1990 ALL RIGHTS RESERVED.
**
**
** Permission to use, distribute, and copy is granted ONLY
** for research purposes, provided that this notice is
** displayed and the author is acknowledged.
**
** This software is provided in the hope that it will be
** useful. BUT, in no event will the authors or Rensselaer
** be liable for any damages whatsoever, including any lost
** profits, lost monies, business interruption, or other
** special, incidental or consequential damages arising out
** of the use or inability to use (including but not
** limited to loss of data or data being rendered
** inaccurate or losses sustained by third parties or a
** failure of this software to operate) even if the user
** has been advised of the possibility of such damages, or
** for any claim by any other party.
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File: setup_grip.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains the statements needed to define a net
**    identical to the one in ~mittmann/nets/grip.nonames_full_io
```

```
**
** Modification History:
**
*/
#include <varargs.h>
#include "pet.h"
#include "pet2.h"
#include "dumb_dec.h"

#define NUM_TRANS 8

char title[] = {"Gripper"};
extern int socket_arr[];
int number_of_sockets = LOCAL_SOCKS;
int menu[SIZE_MENU];
int num_transitions = NUM_TRANS;
struct place_ptr place_ptr_array[6];
struct transition world[NUM_TRANS];
/* ------------------------------------------------------------
 * make_net()
 * this routine just defines a net identical to the one described
 * in the GreatSPN1.5 file grip.nonames_full_io
 *
 * Arguments:none
 * Returns:nothing
 * Requirements: just about all of the routines linked to it ;-)
 * ------------------------------------------------------------*/

make_net()
{
initialize_transition(dummy,TRAN world[0],2,4);
add_pre_list(&(world[0].pre_places), &place_ptr_array[0],
&place_ptr_array[1],
    NULL);
add_post_list(&(world[0].post_places), &place_ptr_array[2],0,
    NULL);

initialize_transition(dummy,TRAN world[1],2,8);
add_pre_list(&(world[1].pre_places), &place_ptr_array[2],
    NULL);
add_post_list(&(world[1].post_places), &place_ptr_array[3],0,
    NULL);
```

```
initialize_transition(dummy,TRAN world[2],4,8);
add_pre_list(&(world[2].pre_places), &place_ptr_array[2],
    NULL);
add_post_list(&(world[2].post_places), &place_ptr_array[3],0,
    NULL);

initialize_transition(dummy,TRAN world[3],6,8);
add_pre_list(&(world[3].pre_places), &place_ptr_array[2],
    NULL);
add_post_list(&(world[3].post_places), &place_ptr_array[3],0,
    NULL);

initialize_transition(dummy,TRAN world[4],8,8);
add_pre_list(&(world[4].pre_places), &place_ptr_array[2],
    NULL);
add_post_list(&(world[4].post_places), &place_ptr_array[3],0,
    NULL);

initialize_transition(dummy,TRAN world[5],10,8);
add_pre_list(&(world[5].pre_places), &place_ptr_array[2],
    NULL);
add_post_list(&(world[5].post_places), &place_ptr_array[3],0,
    NULL);

initialize_transition(dummy,TRAN world[6],2,12);
add_pre_list(&(world[6].pre_places), &place_ptr_array[3],
&place_ptr_array[4],
    NULL);
add_post_list(&(world[6].post_places), &place_ptr_array[54],
socket_arr[0],
    &place_ptr_array[1],0,
    &place_ptr_array[55],socket_arr[0],NULL);

initialize_transition(dummy,TRAN world[7],14,8);
add_pre_list(&(world[7].pre_places), &place_ptr_array[3],
    NULL);
add_post_list(&(world[7].post_places), &place_ptr_array[2],0,
    NULL);

declare_enab_tokens(&(world[1].menu_requirements), MeaFo, NULL);
declare_enab_tokens(&(world[2].menu_requirements), GoFor, NULL);
declare_enab_tokens(&(world[3].menu_requirements), MeaPo, NULL);
declare_enab_tokens(&(world[4].menu_requirements), GoPos, NULL);
```

```
declare_enab_tokens(&(world[5].menu_requirements), Cross, NULL);
declare_enab_tokens(&(world[6].menu_requirements), Retur, NULL);

place_place(0, 20,2);
place_place(1, 2, 2);
place_place(2, 2 ,6);
place_place(3, 2 ,10);
place_place(4, 20,10);
place_place(54, 20,14);
place_place(55, 20,16);
}
void initialize_marking(){
add_token(1,1);
add_token(4,1);
}
```

B.4 setup_vision.c

This file describes the Petri net for the Vision Coordinator.

```
**
** This software was developed at the facilities of the
** Center for Intelligent Robotic Systems for Space
** Exploration, Troy, New York, thanks to generous project
** funding by NASA.
**
** Package: TokenPasser
**
** File:  setup_vision.c
**
** Written By: Michael Mittmann
**
** Date: 1/30/91
**
** Purpose: The purpose of the package can be found in the file
**    main.c.
**    This file contains the definitions needed to define a vision
**    net identical to the one in ~mittmann/nets/vision3_full_io.
**
** Modification History:
**
*/
#include <varargs.h>
#include "pet.h"
#include "pet2.h"
#include "dumb_dec.h"

#define NUM_TRANS 16

char title[] ={"Vision"};
extern int socket_arr[];
int cmmnd_arm();
int number_of_sockets = LOCAL_SOCKS;
int menu[SIZE_MENU];
int num_transitions = NUM_TRANS;
struct place_ptr place_ptr_array[32];
struct transition world[NUM_TRANS];
/* ------------------------------------------------------------
 * make_net()
 * this routine just defines a net identical to the one described
 * in the GreatSPN1.5 file vision3_full_io
 *
 * Arguments:none
```

```
 * Returns:nothing
 * Requirements: just about all of the routines linked to it ;-)
 * ---------------------------------------------------------*/

make_net()
{
initialize_transition(dummy,TRAN world[0],18,4);
add_pre_list(&(world[0].pre_places), &place_ptr_array[16],
&place_ptr_array[17],
    NULL);
add_post_list(&(world[0].post_places), &place_ptr_array[19],0,
    NULL);

initialize_transition(dummy,TRAN world[1],8,4);
add_pre_list(&(world[1].pre_places), &place_ptr_array[16],
&place_ptr_array[17],
    NULL);
add_post_list(&(world[1].post_places), &place_ptr_array[18],0,
    NULL);

initialize_transition(dummy,TRAN world[2],2,16);
add_pre_list(&(world[2].pre_places), &place_ptr_array[16],
&place_ptr_array[17],
    NULL);
add_post_list(&(world[2].post_places), &place_ptr_array[24],0,
    NULL);

initialize_transition(dummy,TRAN world[3],12,8);
add_pre_list(&(world[3].pre_places), &place_ptr_array[19],
    NULL);
add_post_list(&(world[3].post_places), &place_ptr_array[18],0,
    NULL);

initialize_transition(dummy,TRAN world[4],18,10);
add_pre_list(&(world[4].pre_places), &place_ptr_array[19],
&place_ptr_array[20],
    NULL);
add_post_list(&(world[4].post_places), &place_ptr_array[17],0,
    &place_ptr_array[30],socket_arr[0],
    &place_ptr_array[31],socket_arr[0], NULL);

initialize_transition(cmmnd_arm,TRAN world[5],8,12);
add_pre_list(&(world[5].pre_places), &place_ptr_array[18],
```

```
&place_ptr_array[20],
    NULL);
add_post_list(&(world[5].post_places), &place_ptr_array[22],0,
    &place_ptr_array[30], socket_arr[0],
    &place_ptr_array[31], socket_arr[0], NULL);

initialize_transition(dummy,TRAN world[6],0,0);

initialize_transition(dummy,TRAN world[7],12,12);
add_pre_list(&(world[7].pre_places), &place_ptr_array[16],
    NULL);
add_post_list(&(world[7].post_places), &place_ptr_array[23],0,
    NULL);

initialize_transition(dummy,TRAN world[8],8,16);
add_pre_list(&(world[8].pre_places), &place_ptr_array[22],
&place_ptr_array[23],
    NULL);
add_post_list(&(world[8].post_places), &place_ptr_array[24],0,
    NULL);

initialize_transition(dummy,TRAN world[9],8,20);
add_pre_list(&(world[9].pre_places), &place_ptr_array[24],
&place_ptr_array[28],
    NULL);
add_post_list(&(world[9].post_places), &place_ptr_array[25],0,
    NULL);

initialize_transition(dummy,TRAN world[10],18,20);
add_pre_list(&(world[10].pre_places),
&place_ptr_array[20],&place_ptr_array[22],
     &place_ptr_array[23], NULL);
add_post_list(&(world[10].post_places), &place_ptr_array[17],0,
    &place_ptr_array[30], socket_arr[0],
    &place_ptr_array[31],
    socket_arr[0],NULL);

initialize_transition(dummy,TRAN world[11],8,24);
add_pre_list(&(world[11].pre_places), &place_ptr_array[25],
    NULL);
add_post_list(&(world[11].post_places), &place_ptr_array[26],0,
    NULL);
```

```
initialize_transition(dummy,TRAN world[12],8,28);
add_pre_list(&(world[12].pre_places), &place_ptr_array[26],
    NULL);
add_post_list(&(world[12].post_places), &place_ptr_array[29],0,
    NULL);

initialize_transition(dummy,TRAN world[13],8,32);
add_pre_list(&(world[13].pre_places), &place_ptr_array[20],
    &place_ptr_array[29],NULL);
add_post_list(&(world[13].post_places), &place_ptr_array[17],0,
    &place_ptr_array[28],0,
    &place_ptr_array[30],socket_arr[0],
    &place_ptr_array[31],
    socket_arr[0],NULL);

initialize_transition(dummy,TRAN world[14],4,28);
add_pre_list(&(world[14].pre_places), &place_ptr_array[26],
    NULL);
add_post_list(&(world[14].post_places), &place_ptr_array[27],0,
    &place_ptr_array[18],0, NULL);

initialize_transition(dummy,TRAN world[15],4,32);
add_pre_list(&(world[15].pre_places), &place_ptr_array[27],
    NULL);
add_post_list(&(world[15].post_places), &place_ptr_array[28],0,
    NULL);


declare_enab_tokens(&(world[0].menu_requirements), Look, NULL);
declare_enab_tokens(&(world[1].menu_requirements), CalV, NULL);
declare_enab_tokens(&(world[2].menu_requirements), Find, NULL);
declare_enab_tokens(&(world[4].menu_requirements), Retur, NULL);
declare_enab_tokens(&(world[7].menu_requirements), Conti, NULL);
declare_enab_tokens(&(world[8].menu_requirements), CalV, NULL);
declare_enab_tokens(&(world[10].menu_requirements), Retur, NULL);
declare_enab_tokens(&(world[12].menu_requirements), Retur, NULL);


place_place(16,28,2);
place_place(17,8,2);
place_place(18,8,10);
place_place(19, 18,6);
place_place(20, 28,8);
```

```
place_place(22,8,14);
place_place(23,12,14);
place_place(24,8,18);
place_place(25,8,22);
place_place(26,8,26);
place_place(27,4,30);
place_place(28,4,34);
place_place(29,8,30);
place_place(30, 28,22);
place_place(31, 28,34);
}
void initialize_marking(){
add_token(17,1);
add_token(20,1);
add_token(28,1);
}
int cmmnd_arm()
{
    send_tape(socket_arr[0],AD_TAPE_IMMEDIATE, Conti, Slave,
    NULL);
    return(1);
}
```